Brandenburgische
Technische Universität
Cottbus - Senftenberg

**Faculty of Mathematics,
Natural Sciences and
Computer Science**

Institute of Computer Science

COMPUTER SCIENCE REPORTS

Report 01/14

June 2014

COMPARISON OF STOCHASTIC SIMULATION TOOLS

Aman Sinha

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
ISSN: 1437-7969

Send requests to: BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Aman Sinha,
http://dssz.informatik.tu-cottbus.de

Comparison of Stochastic Simulation tools

Computer Science Reports
01/14
June 2014

Brandenburg University of Technology Cottbus - Senftenberg

Faculty of Mathematics, Natural Sciences and Computer Science

Institute of Computer Science

Computer Science Reports
Brandenburg University of Technology Cottbus - Senftenberg
Institute of Computer Science

Head of Institute:
Prof. Dr. Petra Hofstedt hofstedt@tu-cottbus.de
BTU Cottbus - Senftenberg
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus

Research Groups:	Headed by:
Computer Engineering	Prof. Dr. H. Th. Vierhaus
Computer Network and Communication Systems	Prof. Dr. H. König
Data Structures and Software Dependability	Prof. Dr. M. Heiner
Database and Information Systems	Prof. Dr. I. Schmitt
Programming Languages and Compiler Construction	Prof. Dr. P. Hofstedt
Software and Systems Engineering	Prof. Dr. C. Lewerentz
Theoretical Computer Science	Prof. Dr. K. Meer
Graphics Systems	Prof. Dr. D. Cunningham
Systems	Prof. Dr. R. Kraemer
Distributed Systems and Operating Systems	Prof. Dr. J. Nolte
Internet-Technology	Prof. Dr. G. Wagner

CR Subject Classification (1998): I.6.3, I.6.8, G.4, G.3, D.2.2, D.2.8, D.4.8

Printing and Binding: BTU Cottbus - Senftenberg

ISSN: 1437-7969

Comparison of Stochastic Simulation tools

Internship Report

March 07, 2014 to June 15, 2014

Author:

Aman Sinha

Department of Information Technology,
Indian Institute of Information Technology, Allahabad, India

Supervisors:

Prof. Dr.-Ing Monika Heiner
Dipl. Inf. Christian Rohr

Department of Computer Science,
Brandenburg Technical University, Cottbus, Germany

Thursday 14th August, 2014

Abstract

This report compares some stochastic simulation tools for biochemical reaction networks. The stochastic simulation tools are selected on the basis of some selection criteria. Simulations are performed for the different stochastic simulation tools on different benchmark models. This report gives an overview of how the the comparison is carried out for the chosen tools.

The tools are compared on a common evaluation protocol. The evaluation protocol comprises a set of benchmark models along with the parameters which are provided as input to the tools. The benchmark models are represented as Petri nets and fed in SBML (System Biology Markup Language) to the different tools. Experiments are performed on each tool and the results are recorded. The tools are finally compared based on the comparison criteria.

Keywords: stochastic simulation, biochemical reaction networks, evaluation protocol, benchmarks, SBML.

Contents

List of Figures

Task

Stochastic modelling and simulation is gaining increasing attention in systems biology. Accordingly, there are many software tools available which are used for stochastic simulation in the domain of biochemical reaction networks. Each tool has been developed with specific objectives in mind, and most tools announce to be highly efficient.

In this study, we wish to compare features and performance results of some popular tools available for stochastic simulation and in worldwide use in the systems biology community.

The essential steps which will be taken are:

1. Compiling the list of selection criteria,
2. Compiling the list of tool candidates, with their properties with respect to the selection criteria,
3. Selection of the tools for deeper comparison,
4. Compiling the list of comparison criteria,
5. Compiling the benchmark suite,
6. Designing the technology (scripts),
7. Performing simulations on the chosen benchmark test cases,
8. Finalising the report.

1 Introduction

Background. There are numerous stochastic simulation tools developed for performing simulation on biochemical reaction networks. A large variety of modelling techniques are used to model the biochemical reaction networks such as Boolean networks, Differential equations (ordinary or partial), Petri nets, etc. Petri nets are found to be a suitable representation of these biochemical reaction networks. To gain some basic understanding about Petri net models, please refer [10].

Motivation. Petri nets are a Mathematical Modelling Language which is used to describe distributed systems. They are based on the bipartite graph theory where the nodes signify transitions and places. Petri nets are widely used for modelling of biochemical reaction networks. Before starting this performance comparison, a basic understanding of the Petri net framework has been acquired. At the chair of Data Structures and Software Dependability at Brandenburg University of Technology, Cottbus, Germany, two stochastic simulation tools have been developed, namely Marcie and Snoopy. We want to make a performance comparison of these tools with other stochastic tools.

Outline. There are many stochastic simulation tools which are available for educational purposes and are free of charge. We select some of the available tools on the basis of our selection criteria (which is explained later in this report). After selection of the tools we make a performance comparison of these tools based on certain comparison criteria (which is explained later in this report). We perform simulation on some benchmark models, which are all biochemical reaction network models. Simulations are carried out for each of the selected tools. The results obtained by each tool are compared.

2 Tool Selection

There are many tools available for simulation and in worldwide use in the systems biology community. Among them we will consider only tools which meet the selection criteria.

2.1 Selection Criteria

The decision for the selection of a tool is based on the following factors.

1. *Stochastic simulation tool.* There are different approaches to simulate biochemical reaction networks, such as deterministic or hybrid simulation, but in this report we are only interested in stochastic simulation tools.

2. *Algorithms used by the simulator.* There are a couple of algorithms for stochastic simulation, such as Gillespie's direct method, Gillespie's first reaction method, Gibson and Bruck's next reaction method, Tau-leaping etc. The stochastic simulation tool must support at least the Gillespie algorithm, which is considered to be the very basic one.

3. *Active and maintained tool.* Popular and further developed tools and/or in active use in the community, i.e. we select only those tools which are currently maintained, and we select only tools which have a latest release in 2010 or later.

4. *License type.* The use of the simulation tool must be free of charge.

5. *Support of SBML.* The tools should support SBML, level 2, either directly, or indirectly by connection to a tool supporting SBML.

6. *Implementation language of the tool.* The tools should have C or C++ as its implementation language. We are choosing this criteria because of the relative timings of the algorithm in different implementation languages. For more details please see [19]. We are selecting tools on this criteria, but this criteria could be dropped and tools with other implementation languages could also be potential candidates.

2.2 Tool Candidates

There are many tool candidates which may fit the criteria, but for a start we are concentrating on tools reported at the website of the SBML ((System Biology Markup Language)) community
`http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9`.

Potential further sources for tool candidates are
– `http://systems-biology.org/software/simulation/`,
– `http://www.staff.ncl.ac.uk/d.j.wilkinson/smfsb/2e/index.html`.

The following list reports all candidates from the SBML community website supporting stochastic simulation. To be self-contained, we give for each tool a brief description, which is based on the summaries as found on this website. The links to the tools are provided for each tool as well, if known.

However some tools namely SSC, URDME and Marcie are not mentioned on the SBML community website. SSC and URDME was found by searching on Google search engine. SSC has been included because we are interested in model compilation tools.

Marcie is developed at BTU, Cottbus, Germany so we wish to include it in the comparison.

The list is given in lexicographical order.

BetaWB includes the BetaWB simulator, a stochastic simulator based on an efficient variant of the Gillespie Stochastic Simulation Algorithm (SSA), the BetaWB designer, a graphical editor for developing models and the BetaWB plotter, a tool to analyse the results of a stochastic simulation run.

This tool is not selected because it was not available on the linked website. To be precise, the provided link goes to the COSBI top website.

BIOCHAM uses rule based language for modelling biochemical systems, available at `http://contraintes.inria.fr/biocham/`.

This tool is a potential candidate, but is not selected due to time constraints.

Bionessie is a free, state-of-the-art platform-independent biochemical networks simulation and analysis software environment software. It is developed using Java technology and can run on many platforms that support JRE (Java Runtime Environment 1.5 or higher). It provides a full user-friendly Graphical User Interface (GUI) which allows the user to import, create, edit and export the biochemical models with the SBML (Systems Biology Markup Language) standard.

It is not available on the given website `http://www.bionessie.org/URL`, but is available on `http://disc.brunel.ac.uk/bionessie/`

This tool is not selected because its implementation language is Java, however this can be a potential candidate.

Biorica is a high-level hierarchical modelling framework integrating discrete and continuous multi-scale dynamics. The co-existence of continuous and discrete dynamics is assured by flux connections with the continuous parts of the model. Once connected, these parts of the model act as components that can be queried for the function value,

but also modified, therefore accounting for any trajectory modification induced by discrete parts of the model.

BioRica is developed in Python, thus it requires the installation of a couple of Python-related software components.

This tool is not selected because it was not available for download on its website `http://biorica.gforge.inria.fr/software.html`

ByoDYN includes stochastic simulators: SSA and tau-leap. It also integrates ordinary differential equations (ODEs), including systems with events, rules (differential algebraic equations, DAE) and delays built from a given biological model. It also performs Monte Carlo sampling coupled with cluster analysis and PCA to determine the global shape of the parameter landscape. The program makes use of external software, providing a Python binding schema that allows the user to easily implement new software in the desired calculation protocol.

This tool is not selected for the reason that its website `http://cbbl.imim.es:8080/ByoDyn` is not available.

Cain is an application that performs stochastic and deterministic simulations of chemical reactions. It stores models, simulation parameters, and simulation results in an XML format. The models and simulation parameters can be read from input files or edited within the program. Cain offers a variety of solvers including: Gillespie's direct method, Gillespie's first reaction method, Gibson and Bruck's next reaction method, tau-leaping, hybrid direct/tau-leaping, and ODE integration.

It is available at `http://cain.sourceforge.net/`.

This tool is selected for comparison.

Cell Designer is a structured diagram editor for drawing gene-regulatory and biochemical networks. Networks are drawn based on the process diagram, with graphical notation system. CellDesigner supports simulation and parameter scan by an integration with SBML ODE Solver and Copasi. By using CellDesigner, users can browse and modify existing SBML models with references to existing databases (MIRIAM supported), simulate and view the dynamics through an intuitive graphical interface.

It is available at `http://celldesigner.org/URL`.

This tool is not selected for the comparison study because it uses COPASI for its simulation.

COPASI is a software application for simulation and analysis of biochemical networks and their dynamics. COPASI is a stand-alone program that supports models in the SBML standard and can simulate their

behavior using ODEs or Gillespie's stochastic simulation algorithm; arbitrary discrete events can be included in such simulations. COPASI provides an C++ API with language bindings for Perl, python, R, Java, and Octave and is able to communicate with the Systems Biology Workbench COPASI carries out several analyses of the network and its dynamics and has extensive support for parameter estimation and optimization. COPASI provides means to visualize data in customizable plots, histograms and animations of network diagrams. It is also a very popular tool.

It is available at `http://copasi.org/tiki-view_articles.php`

This tool is selected for comparison.

Cyto-Sim is a Stochastic simulator based on automata theory (P system). Free download. JVM Based.

The linked page could not be found `http://www.cosbi.eu/Rpty_Soft_CytoSim.php`.

it can be found on `http://www.cytosim.org/cytosim/cytosim20/index.html`
This tool is not selected because it is Java based tool.

Dizzy is a Chemical kinetics simulator. SBML Import/Export (L1 Subset). Includes Gillespie, Gibson-Bruck and Tau Leap stochastic and ODE/RK5 deterministic methods.

It is available at `http://magnet.systemsbiology.net/software/Dizzy/`.

This tool is selected for comparison, although it is a very old tool and its latest release was in 2006 we have selected it because it is a very popular tool.

E-cell requires Python, Numpy, GSL, Boost. Command line, scripting, and GUI. Supports ODE/DAE and Gibson-Bruck SSA models. GUI requires Gnome, gnome-python2, and pygtk. GPL with exceptions. SBML import via SBML2EML converter. Limited SBML export via ecellj converter.

The tool is available at `http://dev.e-cell.org/redmine`.

This tool is not selected because there is no documentation and install notes present for this tool.

ESS stands for Exact Stochastic Simulator. Part of the UTK/ORNL BioSPICE tool set which includes the BioSpreadsheet SBML model editor. Requires BioSpice Dashboard.

This tool is not selected because the provided link to the website `http://biocomp.ece.utk.edu/` does not work.

Facile Facile/EasyStoch is a command-line network compiler for systems biology. Facile reads models given in a simple and human-readable textual input format and exports the model in a format for readable by Matlab, Mathematica, Maple, XPP/AUTO. Other tools are supported via SBML export. For stochastic simulations, Facile uses the EasyStoch stochastic simulator. An important feature of EasyStoch that distinguishes it from other Gillespie-algorithm implementations is that it is capable of simulating dynamically changing or noisy biochemical parameters (i.e. extrinsic noise). The Facile application is written in the Perl programming language.

It is available at `http://sourceforge.net/apps/mediawiki/facile/index.php?title=Facile/EasyStoch`.

This tool is a network compiler, so it might be interesting. But for the time being it is not selected because of its implementation language and time constraints. However this can be a potential candidate

FERN is developed at LMU, Munich, Germany. It is a Java framework for the efficient simulation of chemical reaction networks. It provides a broad range of efficient and accurate algorithms both for exact and approximate stochastic simulation and a simple interface for extending to new algorithms. Furthermore, it can be used in a straightforward way both as a stand-alone program and within new systems biology applications.

It is available at `http://www.bio.ifi.lmu.de/FERN`.

This tool is not chosen because it is implemented in Java. However this tool can be a potential candidate.

ibioSim supports the modeling, analysis, and design of genetic circuits with applications in both systems and synthetic biology. It includes editors to construct genetic circuit models (GCM), Systems Biology Markup Language (SBML) models (L2V4 and L3V1 supported), and labelled Petri net (LPN) models. Models can be constructed by hand, imported from model databases, or learned from experimental data. These models can be analysed using a variety of ODE and stochastic simulators as well as Markov chain analysis. The efficiency of these analysis methods is enhanced using a variety of automatic reaction-based and logical abstractions. ibioSim is implemented in Java.

This tool is available at `http://www.async.ece.utah.edu/iBioSim/`

This tool is not chosen because its implementation language is Java. However this can be a potential candidate.

Marcie stands for (M)odel checking (A)nd (R)eachability analysis done effi(CIE)ntly Marcie is a tool for qualitative and quantitative analysis

of Generalized Stochastic Petri nets with extended arcs. It comprises four engines.

- Qualitative analysis based on Interval Decision Diagrams (IDD)
- Quantitative analysis based on symbolic exact numerical analysis
- Quantitative analysis based on explicit approximative numerical analysis
- Quantitative analysis based on simulation

This tool is a command line tool and it is available at `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie#description`

This tool is selected for comparison.

MEsoRD is a stochastic and deterministic simulator of coupled chemical reactions and diffusions in space. In particular, it is an implementation of the Next Subvolume Method, which is an exact method to simulate the Markov process corresponding to the reaction-diffusion master equation.

It is available at `http://mesord.sourceforge.net/index.php`.

This tool was not selected because from its overview and documentation it seems that it does not support Gillespie algorithm.

Metaboflux is a computational tool for predicting flux distribution in metabolic networks under multiple and various constraints deducted from the experiments. It aims to increase the biological relevance of models by integrating experimental data. The tool is available in two versions : a command line tool optimized for running on HPC servers and a user-friendly interface designed to define model parameters and run simple computations. Metaboflux encloses a stochastic simulator of metabolic networks coupled with a non linear solver (GSL).

The tool is not selected because the given link to `http://www.cbib.u-bordeaux2.fr/metaboflux/` does not work.

Modesto is a Merged ODE and Stochastic Simulator. Source code only.Linux.

The tool was not selected because it is not available on its web page `http://bioinformatics.oxfordjournals.org/content/20/3/316.abstract` The link goes to a journal paper. However it can be found on
`http://sourceforge.net/projects/modestosim/?source=navbar`

This is not selected because it seems like it was last modified in 2005 and can be considered outdated.

Moleculizer is a command line stochastic simulator. Open source. (Linux). The given link `http://www.molsci.org/~lok/moleculizer/` does not work. However it can be found on `http://moleculizer.soft112.com/`.

The tool is not selected because the latest release was in 2009 so it seems that it is no more maintained. Further this tool also seems unpopular.

sbw:stochastic simulator has GillespieGUI which is a new user interface for biochemical networks that has been designed to integrate both tasks of interest to biologists - namely, simulating a model and analysing the data. The data simulation is carried out by a stochastic simulator, whose parameters such as simulation start and end times, as well as data or time sampling options can be set prior to starting the simulation. This being a tool designed for statistical analysis, users can specify the number of runs of the model that the simulation should generate. Once the data for the specified number of runs has been generated, the tool then computes correlations between various species, along with their power spectral densities and transfer functions.

It is available at `http://jdesigner.sourceforge.net/Site/Stochastic_Simulation.html`

This tool is not selected because it seems not to be popular.

Snoopy is a software tool to design and animate hierarchical graphs, among others Petri nets. To investigate biomolecular networks, Snoopy provides a unifying Petri net framework comprising a family of related Petri net classes. Models can be hierarchically structured, allowing for the mastering of larger networks. To move easily between the qualitative, stochastic and continuous modelling paradigms, models can be converted into each other. We get models sharing structure, but being specialized by their kinetic information. The analysis and iterative reverse engineering of biomolecular networks is supported by the simultaneous use of several Petri net classes, while the GUI adapts dynamically to the active one. Built-in animation and simulation are complemented by exports to various analysis tools.

It is available at `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`

This tool is selected for the comparison.

SSC stands for Stochastic Simulation Compiler (SSC) is a tool for creating exact stochastic simulations of biochemical reaction networks. The models are written in a succinct, intuitive format, where reactions are specified with patterns. Patterns mention only the part of the compound relevant to a given reaction, and correspond to an intuitive

view of biochemical reactions. This enables complex biochemical signalling networks to be specified without the knowledge of any formal programming languages. This tool has been selected for comparison study.It has a special feature of model compilation. This tool has been found by performing Google search.

The tool is available at `http://web.mit.edu/irc/ssc/`.

This tool does not support SBML but due to its claim that it is highly efficient by modifying the Gillespie algorithm we have included it in our comparison study.

STEPS is a package for exact stochastic simulation of reaction-diffusion systems in arbitrarily complex 3D geometries. The core simulation algorithm is an implementation of Gillespie's SSA, extended to deal with diffusion of molecules over the elements of a 3D tetrahedral mesh. Tetrahedral meshes offer much better morphological resolution than the cubic voxels used in other SSA based software and so STEPS is the first spatial SSA software to allow realistic boundary representation. While STEPS was mainly developed for simulating detailed models of neuronal signalling pathways in dendrites and around synapses, it is a general tool and can be used for studying any biochemical pathway in which spatial gradients and morphology are thought to play a role.

This tool is not selected because the given link `http://steps.sourceforge.net/STEPS/` does not work.

StochKit is an extensible stochastic simulation framework developed in C++ that aims to make stochastic simulation accessible to practising biologists and chemists, while remaining open to extension via new stochastic and multi scale algorithms. C++ library that provides various SSA, tau-leaping and adaptive step size algorithms. Source code only. (Linux) (Acad/NP).

It is available at `http://www.engineering.ucsb.edu/~cse/StochKit/`.

This tool is selected for the comparison study. One of the main reason for selecting this tool is its usage and popularity.

StochPy stands for Stochastic modeling in Python is an easy-to-use package, which provides several stochastic simulation algorithms (SSAs), which can be used to simulate biochemical systems in a stochastic manner. Further, several unique and easy-to-use analysis techniques are provided by StochPy including the determination of waiting times.

It is available at `http://stompy.sourceforge.net/`.

This tool has been selected for comparison. Although it is implemented in python we select this tool because is an active and popular tool.

StochSim is a stochastic simulator for (bio)chemical reactions. The particles are represented as individual software objects which react according to probabilities derived from concentrations and rate constants. In the version 1.4 of STOCHSIM simple two-dimensional spatial structures have been implemented, in which nearest-neighbour interactions of molecules can be simulated. This tool seems to aim at simulations in space.

This tool is not selected because it is not available on the website `http://www.pdn.cam.ac.uk/groups/comp-cell/index.html/StochSim.html`

STOCKS is a Stochastic simulation tool which uses SSA, Gibson-Bruck, and Tau-Leaping algorithms. Command line based.

It is available at`http://www.sysbio.pl/stocks/`

The tool is not selected because it was developed in 2004 and since then no progress has been made. So the tool might be considered as outdated.

SynBioSS is a suite of software tools for the modelling and simulation of synthetic gene constructs. SynBioSS utilizes the registry of standard biological parts, a database of kinetic parameters, and both graphical and command-line interfaces to multi scale (stochastic-discrete, stochastic-continuous and continuous-deterministic simulation) algorithms. It is implemented in Perl, Python and PHP.

It is available at `http://synbioss.sourceforge.net/`.

This tool is not selected because of its implementation language.

URDME is a general software framework for modelling and simulation of stochastic reaction-diffusion processes on unstructured, tetrahedral (3D) and triangular (2D) meshes. Unstructured meshes allow for a more flexible handling of complex geometries compared to structured, Cartesian meshes. The current core simulation algorithm is based on the mesoscopic reaction-diffusion master equation (RDME) model.

This tool is not selected because it does not support SBML, it is available at `http://sourceforge.net/apps/trac/urdme/`.

V Cell – The Virtual Cell is a complete model building, editing and simulation environment. Includes spatial modelling capabilities, deterministic, stochastic, and hybrid algorithms. Parameter sensitivity analysis and parameter optimization. Desktop application and web-based environment.

It is available at `http://www.vcell.org/`.

This tool is a potential candidate and can be selected, but for the time being – due to our time constraints – we are not including this tool in our comparison.

2.3 Shortlist

In summary, the tools selected from this list of 31 tool candidates are (ordered lexicographically):

1. CAIN
   California Institute of Technology, Pasadena, California, United States.
   `http://cain.sourceforge.net/`

2. COPASI
   international collaboration between three groups at the Virginia Bioinformatics Institute, the University of Heidelberg, and the University of Manchester
   `http://copasi.org`

3. Dizzy
   Institute for System Biology, Seattle, Washington, US
   `http://magnet.systemsbiology.net/software/Dizzy/`

4. Marcie
   BTU Cottbus, Germany
   `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie`

5. Snoopy
   BTU Cottbus, Germany
   `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`

6. SSC
   MIT, Cambridge, Massachusetts, US
   `http://web.mit.edu/irc/ssc/`

7. StochKit
   UC Santa Barbara University of California, US
   `http://sourceforge.net/projects/stochkit/`

8. StochPy
   VU University Amsterdam, Netherlands
   `http://stochpy.sourceforge.net/`

All selected tools meet the selection criteria.

In the following we report our experience in using these shortlisted tools and we give the reason for all tools which were originally selected, but don't show up in the actual comparison.

Support of SBML. All of the selected tools support SBML except for SSC and Marcie.

SSC is taken into account because it claims that it compiles the model into fast simulator. Part of the speed up comes from algorithmic improvements to the original Gillespie algorithm, while the rest comes from directly generating efficient native code.

Marcie does not directly support SBML, but it is connected to Snoopy, which does support SBML. This tool is chosen for comparison because it is a tool for qualitative and quantitative analysis of generalised stochastic Petri nets with extended arcs. Also, this tool is developed at the hosting institute, BTU, Cottbus, Germany.

Drop outs. From this list of originally selected tools, the following tools dropped out while performing the comparison study.

1. Dizzy: This tool was selected because of its popularity. It supports SBML level 1 version 2, which can be treated as outdated.

   The tool was installed and simulations were performed on the example files provided by Dizzy. However when we tried to import SBML level 2 for the ERK benchmark, it showed an error. An export was developed for SBML level 1 version 2 in Snoopy which was successful. The command line interface was selected for Dizzy. Problems faced while performing simulation on Dizzy were:

   - Dizzy does not display the simulation run time. Experiments were performed on Dizzy for ERK benchmark and the simulation run time was recorded by the time command (Linux time command).
   - It does not support multi threading.
   - The tool is very slow in performing simulation. The experiments were carried out and most of the experiments had simulation runtime greater than 3,600 sec.
   - Dizzy GUI does not support plots for more than 20 species at an instance.

   For more information about experiments with Dizzy please refer Appendix.

2. SSC: This tool was selected because it had special feature of model compilation.

However SSC does not support SBML. We tried to search for a translator for SSC and we found BioNetGen to SSC translator available on `http://bionetgen.org/index.php/BioNetGen_to_SSC_translator`. It says that the new version of BioNetGen (version 2.1.6 and above) have this translator along with themselves. BioNetGen converts a bionetgen file (.bngl) to ssc readable reaction file (.rxn).

The tool was downloaded and installed. The installation steps were followed which were written in the readme file. The tool was successfully installed but we could not get the converter in the BioNetGen directory. Moreover, the steps for translation from BioNetGen file to SSC file is also not provided. This tool is dropped for the reason that it does not supports SBML.

3. COPASI: This tool was selected because it is a popular tool and it satisfies all the selection criteria. In addition it has SBML support. It is able to import SBML level 1 and level 2. Current version supports SBML up to level 2 version 4.

   The tool was installed and it has library dependencies such as Qtlib4. After installation it was found that COPASI does not support averaging. The number of runs cannot be provided at the input terminal. This is the main reason to drop this tool.

4. StochPy: This tool was selected because of it is an active and popular tool. It is a flexible software tool for stochastic simulation in cell biology. It provides various stochastic simulation algorithms, SBML support, analyses of the probability distributions of molecule copy numbers and event waiting times, analyses of stochastic time series, and a range of additional statistical functions and plotting facilities for stochastic simulations. Its latest release is StochPy-1.2.0 which was released in February 2014.
   The link for StochPy can be found at :
   `http://stochpy.sourceforge.net/`
   The tool was downloaded from:
   `http://stochpy.sourceforge.net/download.html`

   After downloading the file was extracted. The installation instructions are given in the README.txt file. The installation steps were followed. The installation requires administrative privileges. The tool flagged an error stating NumPy was not installed.

   StochPy has following dependencies:

   - Python : StochPy has been tested on Python 2.5, 2.6, and 2.7
   - NumPy : StochPy requires NumPy to perform stochastic simulations

- Matplotlib : StochPy requires Matplotlib for plotting

Optional software includes:

- libSBML : StochPy requires libSBML for SBML support
- iPython : Interactive Python Shell
- PySCeS: The Python Simulator for Cellular Systems

However during the installation of Canopy for COPASI, NumPy was installed. StochPy user guide was also referred for installation purpose: `http://stochpy.sourceforge.net/html/userguide.html`

The link given for download of NumPy does not work. Installation of NumPy from the Canopy was only local, but the tool required it to be a system wide installation or global. The package manager of CentOS was used in order to do the installation of NumPy package. This requires administrative privileges.

StochPy ran successfully but while performing SBML import, we need to convert the file into StochPy readable file. The converter is provided by Stochpy itself. When we used the converter there were issues of python bindings.

The installation was found to be difficult because of the dependencies. We decided to drop this tool because of its python binding issues.

After dropping out the tools just discussed, the final list of selected tools in lexicographical order is:

1. CAIN
2. MARCIE
3. Snoopy
4. StochKit

3 Feature Comparison

In this chapter we summarise the main features of those tools which were finally used in the actual performance comparison.

3.1 Comparison Criteria

Comparison criteria can be divided into qualitative and quantitative criteria, which we use in our feature and performance comparison. Our qualitative criteria for feature comparison are:

- **source of origin:**
  name/country of the institute, reference of tool paper, url;
- **modelling paradigm:** stoch, dtm, hybrid; which algorithms?
- **model class:** modelling features beyond standard stochastic models;
- **data exchange formats:** SBML, which version/level any additional formats besides SBML?
- **tool features, handling**
- **interface:** gui/command line tool; screenshot (self-made)
- **evaluation of results:** visualisation, mc, which properties are computed;
- **parallel computing:** multithreading, multiple processes;
- **implementation language:**
- **platforms:** Windows, Linux, Mac OS X,
- **hardware architecture:** 32/64bit
- **license:**
- **tool version:** version number, date of downloading
- **ease of installation:**

Note: Model class which tells about the modelling features beyond stochastic models is beyond the scope of this report. In the following sections, we give qualitative characteristics of all selected tools, which will be summarised in a table afterwards in Section 3.6.

3.2 CAIN

- This tool is developed at California Institute of Technology, Pasadena, California, United States. In order to perform simulation the documentation of CAIN was referred which can be found on `http://cain.sourceforge.net/`. The tool is available for download on `http://cain.sourceforge.net/`.
- Modelling Paradigm: It supports stochastic, deterministic as well as hybrid models. Its simulation method include
  1. Discrete Stochastic Simulations
  2. Direct Method
  3. First Reaction Method
  4. Next Reaction Method
  5. Tau-Leaping
  6. SAL Tau-Leaping
  7. Direct Method with Time-Dependent Propensities
  8. Hybrid Direct/Tau-Leaping
  9. ODE Integration
- Model class: This discussion is beyond the scope of this report.
- Data exchange formats: Imports and Exports
  - It stores models, simulation parameters, and simulation results in an XML format. It supports XML import as well as export.
  - In addition, it also supports SBML imports and exports. The level and versions are not explicitly mentioned in the manual.
  - The results generated can be exported in gnu plot files and it also exports the script for gnu plot to plot the result file.
  - There is a csv export of the simulation result which exports result in the csv format.
- Tool features, handling: The handling of the tool was easy. There are separate panels which make simulation analysis easy. The complete model is described in a single window within their respective panels. E.g. Model Panel, Method Panel, Reaction Panel, Species Panel etc.
- Interface: It is a GUI tool.
- Evaluation of results: CAIN can plot its result by plotting Time Series Data, plotting histograms and tables. It does not support model checking.

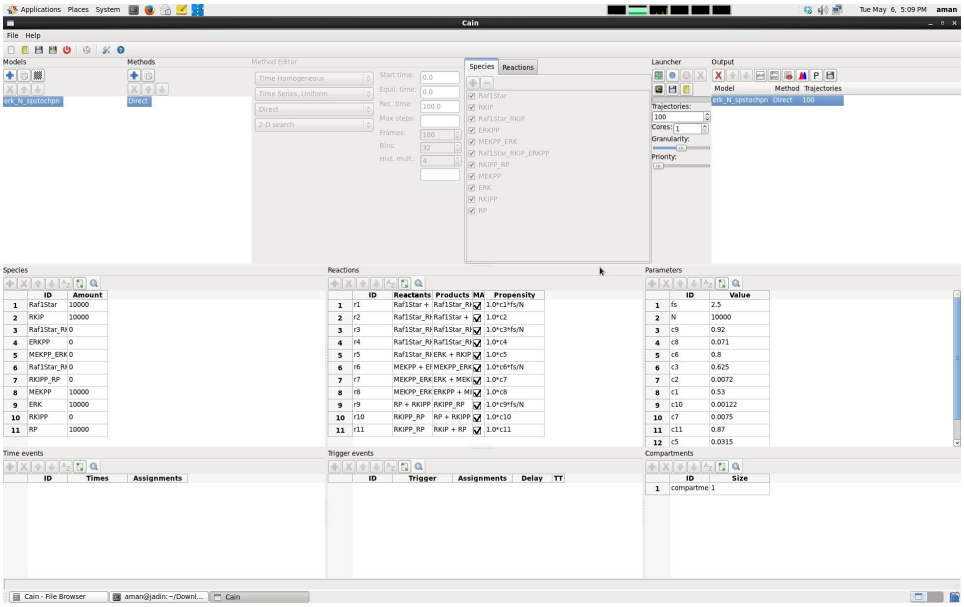

Figure 1: CAIN Screenshot

- Parallel Computing: Yes, implementation principle unknown.
- Implementation Language: The GUI is written in Python and uses the wxPython toolkit. However the solvers are written in C++ and are implemented as command line executables
- Platforms: It supports all the three platforms namely Linux, Windows and Mac/OS.
- Hardware architecture: The 64 bit version was downloaded and installed. The type of architecture is nowhere mentioned exclusively.
- License: Copyright (c) 1999 to the present, California Institute of Technology
- Tool version: The version downloaded was version 1.10. It was downloaded from `http://sourceforge.net/projects/cain/files/cain/` which was made available on sourceforge on 2 July 2012. The website of cain is not updated, it says the latest release is version 1.9 on 27 September 2011. The tool was downloaded on 02 April 2014.
- Easy of installation: The link for CAIN can be found at the SBML website:
  `http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9`

The above link directs to the CAIN website on
http://cain.sourceforge.net/
The download button on the last link will re-direct to
http://sourceforge.net/projects/cain/
from where CAIN can be downloaded. The zip file is downloaded and extracted. The documentation on the cain website was read and steps to install CAIN on REDHAT 6.0/CENTOS 6.0 were followed.
CAIN requires C++ compiler which was already installed on my system. CAIN requires Python, wxPython, matplotlib, numpy and sympy. The easiest way to install the above mentioned package is to install the Enthought Python Distribution. It includes all the packages which CAIN requires. The Enthought Canopy can be downloaded from
https://www.enthought.com/downloads/
The installation guide for Canopy was also read which can be found on
http://docs.enthought.com/canopy/quick-start/install_linux.html
After performing these steps we have sufficient packages installed on the system for CAIN. The CAIN package which was downloaded from
http://sourceforge.net/projects/cain/
was unzipped. The installation instruction for CAIN is available at
http://www.cacr.caltech.edu/~sean/cain/InstallationLinux.htm
The overall installation was easy.

3.3 Marcie

- This tool is developed at Brandenburg Technical University, Cottbus, Germany http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie. In order to perform simulation the user manual was referred. For user manual please refer [16]. The tool is available for download on http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie#downloads

- Modelling paradigm: It is an analysis tool for stochastic petri nets. The engines available are :

  1. Exact Numerical Engine which includes:
     - Jacobi method
     - Gauss-Seidel method
     - Pseudo-Gauss-Seidel method
     - Immediate Transitions
     - Markovian approximation
     - Computation of probability distributions
  2. Approximate Numerical Engine

```
[aman@jadin marcie-linux64-2014-04-17]$ ./marcie

Marcie rev.  (build: ms on 2014-02-17)
A model checker for Generalized Stochastic Petri nets

authors:        Alex Tovchigrechko (IDD package and CTL model checking)

                Martin Schwarick (Symbolic numerical analysis and CSL model chec
king)

                Christian Rohr (Simulative and approximative numerical model che
cking)

                marcie@informatik.tu-cottbus.de

called as: ./marcie


The following options must be additionally specified:
--net-file=   input file (*.apnn, *.andl, *.pnml)



total processing time: 0m0sec

[aman@jadin marcie-linux64-2014-04-17]$
```

Figure 2: Marcie Screenshot

3. Simulative Engine

- Model class: This discussion is beyond the scope of this report.
- Data exchange formats: Imports and Exports
  - MARCIE takes as input the Abstract Net Description Language(ANDL).
  - The file can be created using the ANDL- export feature of Snoopy.
  - It writes simulation result in CSV format.
- Tool features and handling: The tool was found to be easy in handling. The tool is a command line tool and the all the necessary commands which are used while performing simulations are mentioned in the user manual. The results can be exported to a .csv file.
- Interface: It is a command line tool. While simulation, it displays the progress of the simulation (i.e. how much of the simulation is complete).The total processing time includes the simulation run time as well as the time for writing the file. We are concerned with the total elapsed time because it is the simulation run time. The simulation runtime is expressed in the format of 0m0sec.

- Evaluation of results: The simulation results can be exported to a .csv file which can be processed by gnuplot in order to plot the graph. It does not support any plotting function. It support model checking.
- Parallel computing: Yes, implementation principle unknown.
- Implementation language: MARCIE is written in C++.
- Platforms: It is supported in Linux and Mac/OS. Hardware architecture: Only 64 bit for Linux was downloaded and installed. The current version is available for MAC OS 10.5/6 , Linux32 and Linux64.
- License: It's available free of cost for academic purpose.
- Tool version: MARCIE was first released on 23 December 2010. The latest release of MARCIE was on 19 July 2012. The latest release of MARCIE was downloaded and used for performing simulation. The tool was downloaded on 17 April 2014.
- Ease of installation: MARCIE can be downloaded from the link : `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Marcie#download`
  The downloaded file can be extracted and MARCIE can be run directly by going into the sub-folder.
  However MARCIE requires GLIBC version 2.14 and GLIBCXX 3.4.15 for its execution.
  The installation for this tool was found to be easy.

3.4 Snoopy

- This tool is developed at Brandenburg Technical University, Cottbus, Germany `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`. In order to perform simulation please refer [10] and [11]. The user manual was also referred. For user manual please refer [7]. For the graph based data structure used in Snoopy and modeling and simulation in Snoopy refer [6]. The tool is available for download on `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#downloads`
- Modelling paradigm: The available simulators are stochastic, deterministic and hybrid. The algorithms available are :
  1. Stochastic Simulators:
     - Gillespie
     - FAU
  2. Continuos Simulators:

- BDF
- Rosenbrock-Method of Shampine
- Rosenbrock-Method GRK4T of Kaps-Rentrop
- Rosenbrock-Method GRK4A of Kaps-Rentrop
- Rosenbrock-Method of Van Veldhuizen [gamma = 1/2]
- Rosenbrock-Method of Van Veldhuizen [D-stable]
- an L-stable Rosenbrock-Method

3. Hybrid Simulators:
   - Explicit RK
   - Implicit RK
   - BDF
   - ADAMS

- model class: This is beyond the scope of this report.
- Data exchange formats: Imports and Exports
  - It can import as well as SBML level 2 version 3.
  - It supports several other imports and exports. For more imports and exports visit the web page `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#imexport`
- Tool features and handling: The tool was found to be easy in handling. The tool provides a special animation mode where you can play the token game, which helps in better understanding of the model. The simulation window is very easy to handle. The graphs are plotted automatically. The simulation control panel contains the different functions sets, parameters, simulators etc.
- Interface: It is a GUI tool.
- Evaluation of results: The default is the graphical plot which appears on the simulation window. The results can be exported in csv format as well as image can also be exported (e.g. gif, bmp etc). The viewer view panel has three options xy plot, histogram, and tabular. The xy plot shows the graphical lines in the simulation window, the histogram shows the graphical histogram representation in the simulation window and the tabular view shows the result in the tabular format in the simulation window. It does not support model checking.
- Parallel computing: Yes, implementation principle unknown.
- Implementation language: Snoopy is implemented in C++, wxWidgets, Xerces.

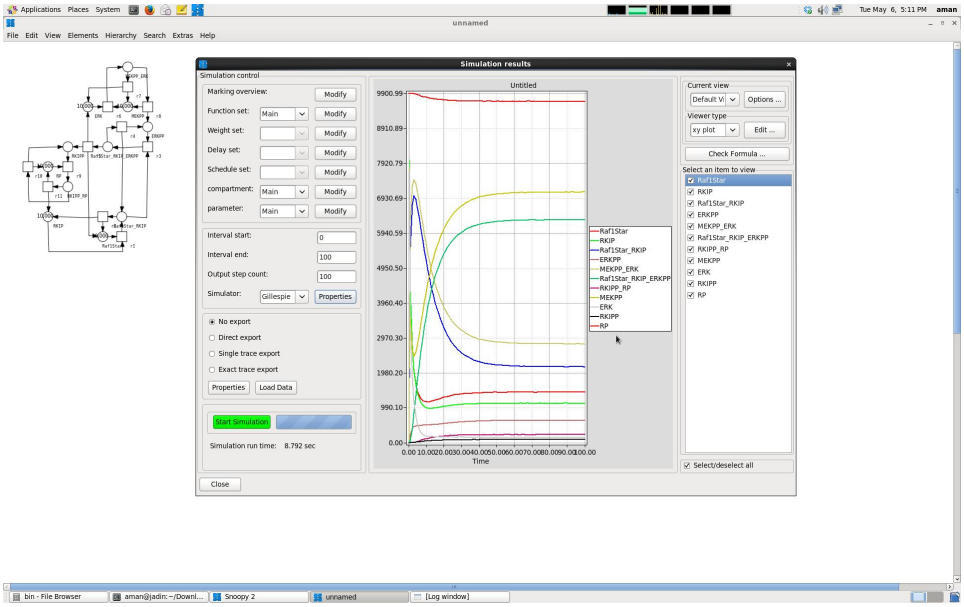

Figure 3: Snoopy Screenshot

- Platforms: It supports all the three platforms namely Linux,Windows and Mac/OS.
- Hardware architecture: Only 64 bit for Linux was downloaded and installed. 32 bit version is available for Linux, however the architecture is not explicitly mentioned for Mac and Windows.
- License: This is available free of cost for academic purpose.
- Tool version: The version downloaded was version 1.13. Snoopy was first released on 9 October 2008. Its latest release was on 01 April 2014. The tool was downloaded on 14 April 2014.
- Ease of installation: The link for SNOOPY can be found at the SBML website:
  `http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9`
  The above link directs to the SNOOPY website on:
  `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy`
  Clicking on the download tab will direct the page to the direct link for the SNOOPY on the same page.
  `http://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Snoopy#downloads`
  The CentOS version of SNOOPY was downloaded. It was extracted.

After extraction, snoopy2.sh file was run. This will run Snoopy. The installation manual for Snoopy is not present.
The installation was found to be easy.

3.5 StochKit

- This tool is developed at UC Santa Barbara University of California, United States. `http://sourceforge.net/projects/stochkit/`. In order to perform simulation the user manual was referred which is provided with the installation file. The tool is available for download on `http://sourceforge.net/projects/stochkit/`

- Modelling paradigm: StochKit2 provides commandline executables for running stochastic simulations using variants of Gillespies Stochastic Simulation Algorithm and Tauleaping. Improved solvers including efficient implementations of :

  1. SSA Direct Method
  2. Optimized Direct Method
  3. Logarithmic Direct Method
  4. ConstantTime Algorithm
  5. Adaptive Explicit Tauleaping method

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - The source file is stored in a .cpp format.
  - Uses a Java Converter to convert the SBML input file to make it compatible with StochKit.
  - The converter accepts the standard version 1 (level 1 and level 2) of SBML and version 2 SBML files.

- Tool features and handling: The tool was found to be easy in handling. The tool is a command line tool and the all the necessary commands which are used while performing simulations are mentioned in the user manual. The results can be exported to a .txt file. it exports means as well as variance of the species in the reaction. It has a special feature of determining the simulation method based on the model that will achieve best performance while simulation. This can be seen in 4. In the second line it states that "StochKit MESSAGE: determining appropriate driver...running '$STOCHKIT_HOME/bin/ssa_direct _small'..."

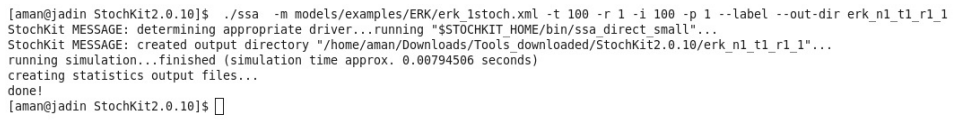

Figure 4: StochKit Screenshot

- Interface: It is a command line tool. It displays the drivers which it uses while performing simulation. The simulation runtime is displayed at the end.
- Evaluation of results: The simulation results can be exported to a .txt file which can be processed by gnuplot in order to plot the graph. It supports plotting function. The plotting tools are available in MATLAB. It does not support model checking.
- Parallel computing: Yes, implementation principle unknown.
- Implementation language: Stochkit is written in C++.
- Platforms: It supports all the three platforms namely Linux, Windows and Mac/OS.
- Hardware architecture: Only 64 bit for Linux was downloaded and installed. There is no explicit mention of the architecture.
- License: StochKit2 (version 2.0.5 and later) is distributed under the BSD 3Clause License (BSD New or BSD Simplified).
- Tool version: The latest release of StochKit is StochKit 2.0.10 on 20 November 2013. The latest release of StochKit was downloaded and used for performing simulation. The tool was downloaded on 02 April 2014.
- Ease of installation: The link for StochKit can be found at the SBML website:
  `http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#cat_9`
  The above link directs to the StochKit website on:
  `http://www.engineering.ucsb.edu/~cse/StochKit/`
  The above link will be directed to sourceforge for the download option:
  `http://sourceforge.net/projects/stochkit/`
  StochKit2 was downloaded and extracted. In the extracted folder there is a StochKit2 manual. The installation steps written in the manual were followed. StochKit was installed successfully.

However for importing SBML files we need SBML converter. The SBML converter was found in the tools sub-folder. The documentation was read and the steps to install the SBML converter were followed. It needs an additional library libSBML which needs to be installed. For installation of LibSBML at standard location administrative privileges are required. The LibSBML converter did not install successfully.
The current system was installed with libSBML version 4.7. However libSBML version 4.1.0 was installed as stated in the documentation manual of SBML converter. The installation was local to the system. So we needed to specify the path. After performing the steps written in the documentation file, the SBML converter was installed successfully.
The installation was found to be difficult.

3.6 Summary

The qualitative analysis of the selected tools for comparison study is summarized in Table 1.

	CAIN	MARCIE	SNOOPY	STOCHKIT
Gillispie Method	Yes	Yes	Yes	Yes
SBML Support (Level 2)	Yes	Yes, but through Snoopy	Yes	Yes, using converter
Handling	Easy	Easy	Easy	Easy
Interface	GUI	Command Line	GUI	Command Line
Output File format	.csv	.csv	.csv	.txt
Plot Function	Yes	No	Yes	Yes, through MATLAB
Parallel Computing	Yes	Yes	Yes	Yes
Implementation lang.	C++, python, wxPython	C++	C++, wxWidgets, Xerces	C++
Platforms	Linux, Windows, Mac OS X	Linux, Mac OS X	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X
Ease of Installation	Easy	Easy	Easy	Difficult

Table 1: Qualitative Comparison of tools

4 The Benchmark Suite

Selection criteria. The models selected for comparison shall fulfil the following properties.

- *Standard Petri net.* The model does not use any special Petri net modelling features, such as read arc, inhibitor arc, immediate transitions, deterministic transitions, etc.
- *Scalability.* There should be at least one model parameter for scaling the marking and/or net structure.

Model form. The following sections summarise all benchmark examples; the information is structured into:

- *Description.* a brief description of the example including a figure showing the Petri net model, and some references where it has been published.
- *Scaling parameter.* List of parameters and their meaning for model scaling.
- *Model size.* Size of the Petri net model in terms of number of places, transitions and arcs. These numbers have been found by importing the SBML file in Snoopy and viewing the net information.
- *Simulation parameters.* Chosen setting for the simulations, such as interval start time, interval end time, interval steps, value of scalable parameter, number of runs, number of experiments per run and number of threads.

4.1 ERK Model

Description. The RKIP inhibited ERK pathway was originally published in [8], and discussed as qualitative and continuous Petri nets in [2], and as three related Petri net models comprising the qualitative, stochastic and continuous paradigms in [4], see Figure 5.

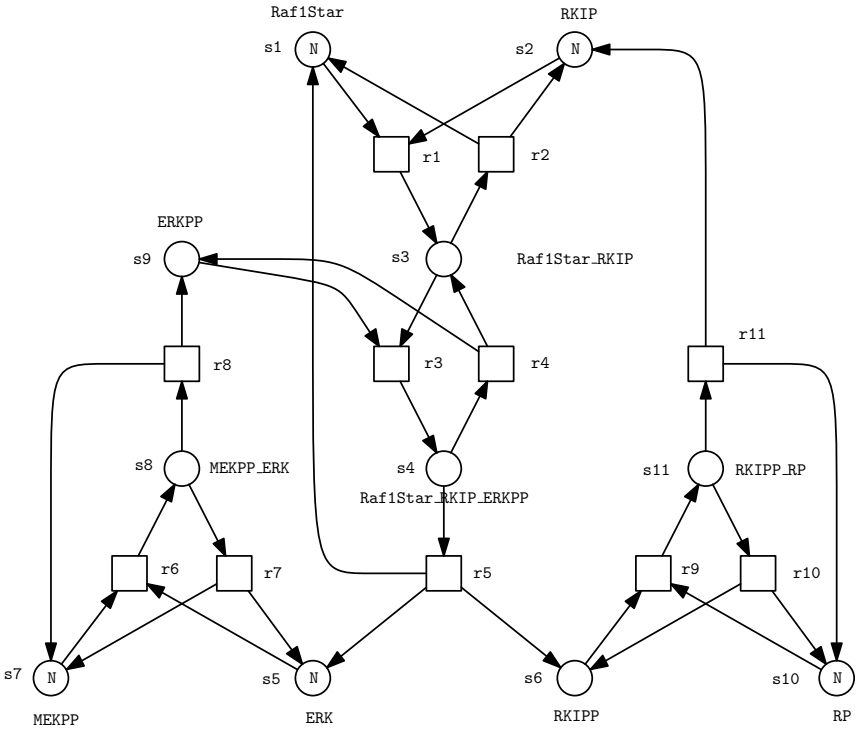

Figure 5: Petri net representation of the ERK model.

Scaling parameter

- N – the initial number of tokens on the places ERK, MEKPP, Raf1Star, RKIP and RP;

Model size

- number of places: 11
- number of transitions: 11
- number of arcs: 34

Although the model is parametrized, the size of its structure does not depend on the parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 100, 10,000, 1,000,000
- no of runs: 1, 100, 10,000, 1,000,000
- no of experiments per run: 10
- no of threads: 1, 4

4.2 LEVCHENKO Model

Description. The mitogen-activated protein kinase (MAPK) cascade was published in [9]. This is the core of the ubiquitous ERK/MAPK pathway that can, for example, convey cell division and differentiation signals from the cell membrane to the nucleus. It has been used in [3] and [5] as running example to discuss three related Petri net models comprising the qualitative, stochastic and continuous paradigm, see Figure 6.

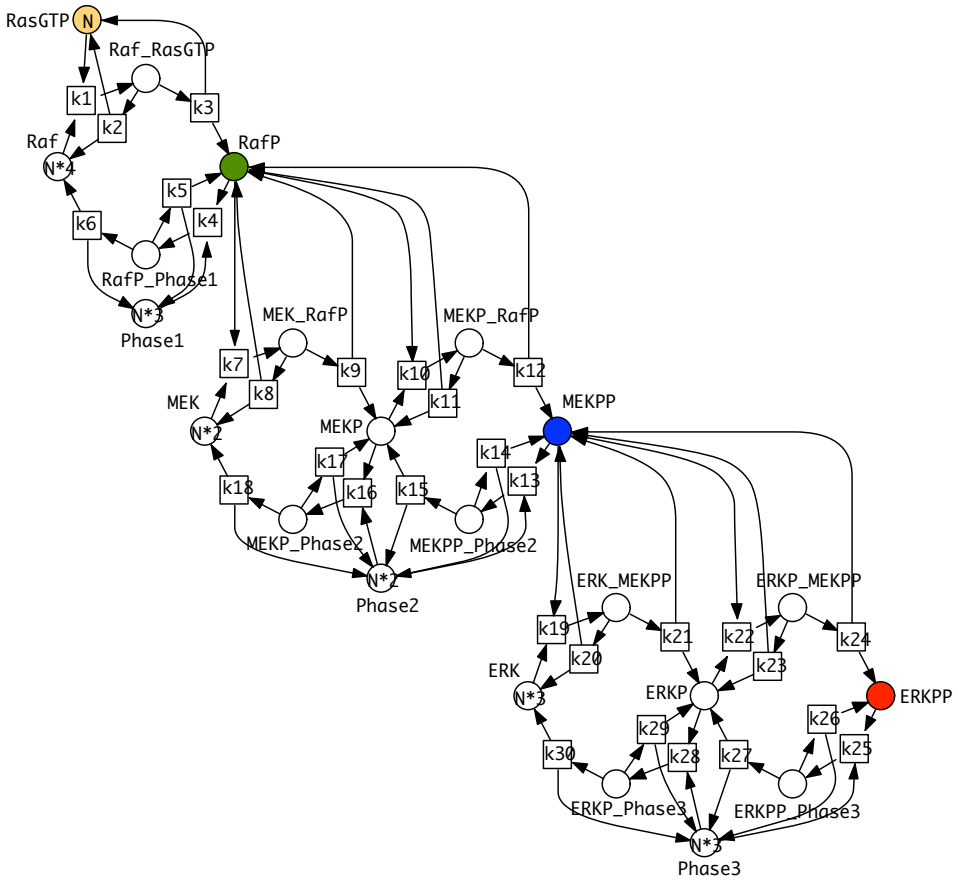

Figure 6: Petri net representation of the LEVCHENKO model.

Scaling parameter

- N – the multiplier of the initial number of tokens on the places Raf, RasGTP, RafP_Phase1, MEKP_Phase2, ERk, ERKP_Phase3

Model size

- number of places: 22

- number of transitions: 30
- number of arcs: 90

Although the model is parametrised, the size of its structure does not depend on parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 10, 100, 1,000
- no of runs: 1, 100, 10,000, 1,000,000
- no of experiments per run: 10
- no of threads: 1, 4

4.3 ANGIOGENESIS Model

Description. Angiogenesis, defined as the formation of new vessels from existing ones, is a topic of great interest in all areas of human biology, particularly to scientists studying vascular development, vascular malformation and cancer biology. Angiogenesis is a complex process involving the activities of many growth factors and relative receptors, which trigger several signalling pathways resulting in different cellular responses. The Petri net was introduced in [13] and refined in [1], see Figure 7.

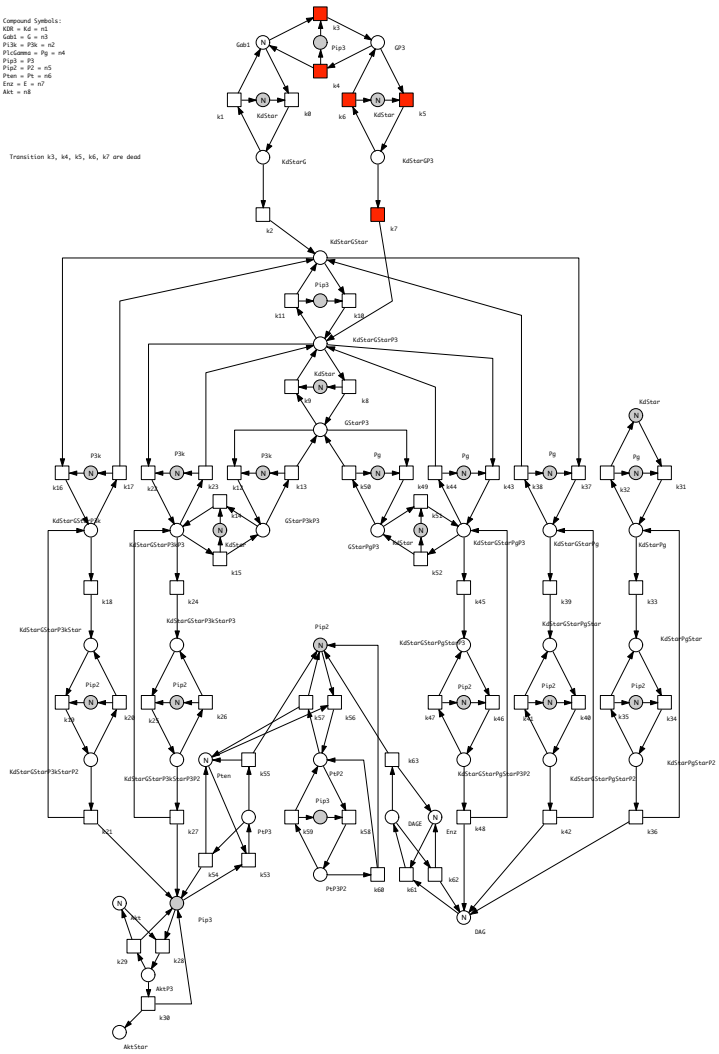

Figure 7: Petri net representation of the ANGIOGENESIS model.

Scaling parameter

- N – initial number of tokens on places Akt, Enz, Gab1, KdStar, P3k, Pg, Pip2 and Pten

Model size

- number of places: 39
- number of transitions: 64
- number of arcs: 185

Although the model is parametrized, the size of its structure does not depend on parameter values.

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- value of N: 1, 5, 10, 50
- no of runs: 1, 100, 10,000, 1,000,000
- no of experiments per run: 10
- no of threads: 1, 4

4.4 CIRCADIAN CLOCK Model

Description. The abstract circadian clock model of Barkei and Leiber [12] shows circadian rhythms which are widely used in organisms to keep a sense of daily time. The stochastic Petri net of the circadian clock is based on the ODE model of [18]. The bounded version of the net was used in [17] and the unbounded version in [14]. Here we use the unbounded version which is the original version, see Figure 8. It unbounded version has no scalable parameter. We choose the Circadian Clock, because it has no upper bound on the number of tokens, whereas the other models have an upper bound in the number of tokens.
Therefore the Circadian Clock has no scalability property. We wanted to see, how the tools would work with such a model.

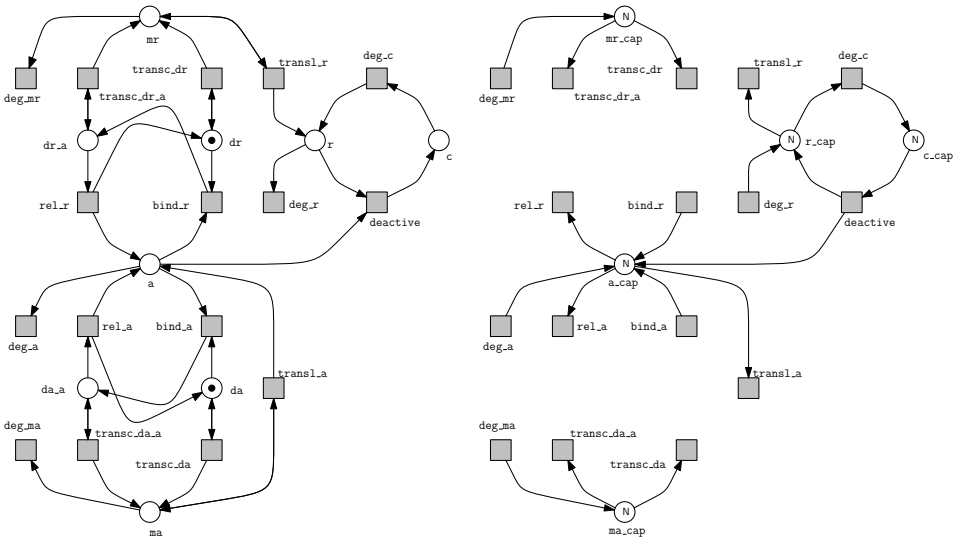

Figure 8: Petri net representation of the CIRCADIAN CLOCK model.

Scaling parameter

- The unbounded version of Circadian Clock model has no scaling parameter.

Model size

- number of places: 9
- number of transitions: 16
- number of arcs: 39

Simulation parameters

- interval start time: 0
- interval end time: 100
- interval steps: 100
- no of runs: 1, 100, 10,000, 1,000,000
- no of experiments per run: 10
- no of threads: 1, 4

5 Performance Comparison

System configuration details:

- Hardware:
  - Workstation: Dell Precision T7400
  - RAM: 4x1024 MB DDR2 FB-DIMM @ 667MHz
  - Processor : Intel (R) Xeon (R) CPU E5440 @ 2.83 GHz
  - CPU cores: 4
  - Cache size: 6,144 KB
  - Cache alignment: 64
- Software:
  - Operating System : CentOS release 6.5 (64bit)

5.1 Comparison Criteria

The comparison criteria is based on the following parameters:

1. Accuracy
2. Simulation runtime comparison
3. Memory consumption comparison
4. Multi-threading Coefficient

The assumptions and constraints while performing simulation are:

- We are interested in the mean value of the species.
- The simulation algorithm used is Direct/ Gillespie.
- Threads used will be either 1 or 4 which will be mentioned explicitly.
- We define an experiment as simulation carried out for a particular value of N, number of runs and threads.
- A total of 10 trials is performed for each experiment and for each benchmark model.
- A script for calculating memory consumption is used.
- After performing 10 trials, the mean value of the simulation runtime and the corresponding peak memory consumption is calculated.

- The threshold simulation runtime for a particular model is 3,600 seconds (1 hour). If the simulation runtime for a particular model is more than 3,600 seconds (1 hour) then we terminate the simulation.
- Simulation runtime which is recorded is the runtime displayed by the tool after completion of the simulation. The simulation runtime is rounded off up to 3 places of decimal. The average runtime is rounded up to 4 places of decimal.
- The memory is calculated using a shell script. It calculates memory consumption in KB and has a sampling time of 0.1 seconds.
- For a GUI tool in-order to calculate the memory consumption the tool has to be reopened before each experiment.

Following steps are followed while performing each experiment:-

- 10 trials are carried out keeping the scaling parameter, no of runs and no of threads constant.
- Simulation runtime for each trial is noted.
- The memory consumption for each trial is also recorded and the maximum/peak memory consumption is taken into account.
- The runtime of each trial is recorded. Such 10 trails are recorded and the average runtime is calculated. The average runtime calculated is the runtime of a particular experiment.
- This average runtime and the peak memory consumption is the simulation runtime and memory consumption of an experiment respectively.
- While performing simulation on CAIN the granularity and priority sliders are kept to their default value.
- While performing simulation on Marcie only the total elapsed time is noted. The total elapsed time is the runtime of the simulation.
- Stochkit uses a SSA driver for performing stochastic simulation. It selects appropriate simulation method to achieve the best performance. For more details see [15] and refer StochKit manual.

The runtime of the tools are interpreted as:

Tools	Steps			
	Read	Simulate	Plot	Write
Snoopy	NO	YES	YES	NO
Marcie	YES	YES	NO	YES
StochKit	NO	YES	NO	NO
CAIN	NO	YES	NO	NO

Table 2: Runtime interpretation of the tools.

The above table means that the simulation runtime of a particular tool is determined by the above steps. e.g. simulation runtime for snoopy includes the simulation time of the direct/gillespie algorithm as well as plotting of the curve. It doesn't include the reading time of the SBML file and the time spend in writing the result into a file (in this case .csv file).

In case of Marcie the total processing time includes time for reading, simulation and writing. However we are only interested in the time for simulation. The simulation time displayed in Marcie is the total elapsed time. We record the total elapsed time for the experiments.

Multithreading Coefficient. Threading coefficient of an experiment is calculated by dividing the simulation runtime of thread 1 by simulation runtime of thread 4. We only take out threading coefficient for those experiments in which the simulation runtime is greater than 10 sec when performed on thread value equal to 4. Here we denote the threading coefficient term by β, defined as

$$\beta = \frac{simulation\_runtime\_of\_thread1}{simulation\_runtime\_of\_thread4}.$$

The table for threading coefficient for each tool is provided. The threading coefficient should ideally lie in the range between 1 and 4 i.e. $1 \leq \beta \geq 4$. Average threading coefficient is calculated for each tool corresponding to each benchmark model. This is calculated by taking the average of the all the β values. Threading coefficient is rounded off to four places of decimal. Variation for threading coefficient is equal to the standard deviation.

The graph plots are log-scaled. Since we take average simulation runtime of a tool, if a tool has a runtime of 0 sec, in order to plot the graph (since log(0) is not defined) we take the next higher value of the experiment and normalize it with the given run. For e.g. for Marcie for the experiment ERK model at N=1 thread=1 and run=1 refer marcie table 7 we calculate the runtime by dividing the runtime at N=1 T=1 and R=1,000,000 by corresponding no of runs (i.e. 1,000,000) and multiplying with the no of runs of the chosen experiment (i.e. 13/1,000,000 * 1). We round off this value up to 4 places of decimal. If the value still comes out to be 0.0000 then we consider it as 0.0001. The table for the average runtime is not affected by this calculation. This is done just for the sake of convenience in plotting the

graphs.
The runtime plots contain '+' sign in brown colour for Marcie. Before the '+' sign (as well as including the '+' value) all the values are approximated using the assumption mentioned above.

Note 1: Marcie displays its simulation runtime in seconds with no significant digits. So the average value of the simulation runtime is calculated up to 1 place of decimal only. The table entry has memory consumption of 0 KB which is not possible. Since the sampling time of the script is 0.1 seconds, for simulation in Marcie which gets over before 0.1 seconds, the script is not able to calculate the memory consumption.

Note 2: Peak in the memory consumption plot is due to the script. The plot should be a straight line with slope ≈ 0. For small simulation runtime the memory consumption cannot be recorded accurately by the script. The peak observed is due to the sampling rate of the script. The sampling rate of the script is 0.1 seconds. So for smaller runtime the script may give a peak. We cannot change the sampling rate of the script because it will affect the simulation runtime of the tool.

Convention for tool comparison plots:

- *Runtime.*
  - For scalable models: For a particular value of thread and no of runs we plot the graph between the simulation runtime in seconds (on y axis) and scaling parameter (on x axis). The graph is log scaled.
  - For non scalable model (e.g. Circadian Clock) we plot the graph for a particular no of thread. The y axis represents the runtime in sec and the x axis represents number of runs. The graph is log scaled on both the axes.
- *Memory Consumption.*
  - For scalable models: For a particular value of thread and no of runs we plot the graph between the memory consumption in KB (on y axis) and scaling parameter (on x axis). The graph is log scaled.
  - For non scalable model (e.g. Circadian Clock) we plot the graph for a particular no of thread. The y axis represents the memory consumption in KB and the x axis represents number of runs. The graph is log scaled on both the axes.
- *Relative comparison of tools based on:*

1. Runtime:
   - On x axis we have the scaling parameter for benchmarks having scalable parameters. If the benchmark doesn't have any scalable parameter (e.g. Circadian Clock) we take runs on the x-axis. On y axis we have the relative runtime in sec.
   - In order to make relative comparison of tools, we select the tool which has the maximum runtime. We call this tool as the base tool.
   - We choose a particular value of runs at which all the tools have significant value of runtime.
   - The runtime of each tool is subtracted from the runtime of the slowest tool. The positive value shows by how much value the tool is faster than the base tool.
   - The negative value show by how much value the tool is slower than the base tool.
   - The graphs are plotted for different value of threads and for each benchmark.
   - The x axis of the graph is log scaled.
2. Multi threading co-efficient:
   - In order to make relative comparison of tools, we select the tool which has the least multi-threading coefficient. We call this tool as the base tool.
   - We enumerate benchmarks as ERK= 1, LEVCHENKO= 2, ANGIOGENESIS= 3 and CIRCADIAN CLOCK= 4.
   - We perform normalization of the multi threading coefficient by diving the multi threading coefficient of each tool by the multi threading coefficient of the base tool.
   - On x axis we have the enumeration of the benchmarks and on y axis we have the normalized multi threading coefficient with respect to the base tool.
   - This calculation and the table generation is done manually.
   - The normalized value shows that by how much factor the tool utilizes the multi threading property in comparison to other tools.

5.2 Technology

Steps to be followed:

1. For GUI Tool:

   - Open GUI tool in the terminal along with memory usage script. e.g. /home/aman/Dropbox/aman/benchmarks/memusg ./snoopy2.sh. The memory script memusg calculates the peak memory usage by any application.
   - Perform simulation on a particular benchmark for a particular value of scaling parameter, thread and run. This is termed as one experiment. Perform 10 trials of each experiment. Note the simulation run-time displayed by the tool on paper, export the traces and follow the standard file naming convention and standard folder convention .
     Standard file naming convention is :
     <tool_name>_<model _initial> _<scaling _parameter _along _with _the _value> _<T _thread _value> _<R _runs _value> _<experiment _no>.<file _extension>
     e.g. SNOOPY_ERK_N1_T1_R1_1.csv means the traces are for SNOOPY, for ERK model, for scalable parameter value = 1, thread value = 1, runs value = 1, trial number = 1, extension = .csv
     Standard folder naming convention is:
     <Model_name> -> <tool_name> -> <scalable_parameter_along_with_value> -> <thread_value> -> <runs_value>
     Note: '->' sign denotes the folder hierarchy.
     e.g. SNOOPY_ERK_N1_T1_R1_1.csv can found by going through the ERK folder, then in SNOOPY sub-folder, then in 'ERK_N1' folder, then in 'threads_1' sub-folder, then in 'runs_1' sub-folder.
   - Close the tool and record its peak memory usage.
   - If simulation runtime > 3,600 seconds. Terminate the simulation.
   - Perform 10 trials for each experiment.
   - A spreadsheet is created manually and the memory consumption and simulation runtime are entered manually. The average runtime and peak memory consumption can be calculated using the functions available in the spreadsheet.

2. For Command Line tools:

   - Shell script for benchmark is created. The benchmark shell script stops the simulation once the simulation runtime is > 3,600 seconds. The shell script stores the output of the terminal for a

particular experiment in a .out file. This shell script calls the memory usage script in order to compute the peak memory consumption. A .csv is created where the memory consumption and the runtime of the tool is noted. This runtime is noted using the time command of Linux. However, we are not interested in this runtime. We are interested in the simulation run time displayed by the tool. Hence to develop a parser for the .out file.

- Once the .csv file containing the memory consumption and runtime and the .out file is created, we parse the .out file and the .csv file through a C# script. This C# script reads the memory consumption from the earlier .csv file (the csv file created by the shell script) and reads the simulation runtime from the .out file. The C# script writes a .csv file which contains the simulation runtime and memory consumption of each experiment. This C# script also writes the average simulation runtime and the peak memory consumption.
- Shell script for StochKit and Marcie is created which calls the benchmark shell script along with the command line syntax for Marcie and StochKit.
  Command line syntax for:
  - Marcie: <marcie_path> –simulative –net-file= <net_file_path> –sim-stop= <sim_stop_time> –sim-out-steps= <no_of_interval_steps> –const <value_of_scalable parameter> –threads= <value_of_thread> –sim-result-file= <output_file_path/output_file_name>
    e.g.
    ```
    marcie \
    --simulative \
    --net-file=erk_N.andl \
    --sim-stop=100 \
    --sim-out-steps=100 \
    --const N=1 \
    --threads=1 \
    --sim-runs=1 \
    --sim-result-file=MARCIE_ERK_N1_T1_R1_1.csv
    ```
    Note: The net file provided to Marcie is in andl format. This .andl file is created by importing the sbml file into Snoopy and then exporting it into andl format.
    For more information on using marcie commands please refer [16].
  - StochKit: <stochkit_driver_name> -m <model_*name* > -t <end_time_interval> -r <no_of_runs> -i <interval_step_count> -p <thread_value> –label –out-dir <output_file_path/output_file_name>
    e.g.

./ssa -m /home/aman/Dropbox/aman/benchmarks/erk/stochkit_erk_1.xml
-t 100 -r 1 -i 100 -p 1 –label –out-dir STOCHKIT_ERK_N1_T1_R1_1
Note: The model file provided for StochKit is the file which has been converted by the StochKit2SBML converter.
SBML converter syntax:
<path_of_SBMLconverter> <path_of_SBML_file> <path_of_converted_file>
e.g.
/usr/bin/time -v /home/aman/BTU_Intern/Scripts/memusg
/home/aman/Downloads/Tools_downloaded/
StochKit2.0.10/tools/SBMLconverter/bin/sbml2stochkit
/home/aman/Downloads/Tools_downloaded/benchmarks/erk/erk_1.xml
/home/aman/Downloads/Tools_downloaded/benchmarks/erk/stochkit_erk_1.xml
Note: The time and the memory usage script is used in order to calculate the time and memory consumed while conversion.
For more information about the StochKit commands please refer the user manual for StochKit.

3. For plotting the graph and making comparison:

   - Runtime graphs are plotted for a specific value of thread and specific value of runs. The x axis denotes the scaling parameter and the y axis denotes the simulation runtime (in sec).
   - Memory consumption graphs are plotted for a specific value of thread and specific value of runs. The x axis denotes the scaling parameter and the y axis denotes the memory consumption (in KB).
   - A separate table for runtime and memory consumption is created similar to 9 and 10 for a specific thread and for each tool. So each table contains 16 entries. A † is represented by a blank field. The file name convention for each table is:
     <tool_name>_<runtime/memory>_<model_name>_chart_t<thread_value>.txt
     e.g.
     cain_runtime_erk_chart_t1.txt means the table is for tool CAIN. The table contains the average simulation runtime of the experiments for model ERK for thread value=1.
   - For a tool whose runtime is approximated (e.g. Marcie) a separate breakpoint table needs to be created which specifies the runtime value after which the runtime values are reported by the tool. The structure of the table is similar to the runtime table created in the previous step.
   - The value in the tables are plotted using the GNU plot scripts.
   - This generates the simulation run-time and the memory consumption plots.

- A beta value table is created manually where we only choose the experiments whose simulation runtime is ≥ 10 sec when performed with thread value= 4. The β value is calculated by dividing the simulation runtime of thread 1 by the simulation runtime of thread 4. The average value of β for each tool is noted and the variation (i.e. standard deviation) is noted. The mean value of beta and standard deviation can be calculated by using the function in spreadsheet application.
  Note: If there is only sample point then the standard deviation is not calculated.

4. Relative comparison of tools based on:

   (a) Runtime:

   - A table similar to Table 39 is made. The table is stored as a .csv file and the data field separator should be TAB. The table entries denotes how much time a tool is faster than the base tool.
   - The positive entries in the table denote by how much value the tool is faster than the base tool.
   - The negative entries in the table denote by how much value the tool is slower than the base tool.
   - A C# script is developed which reads the table entry and creates two tables for each thread.
   - These tables are plotted with the help of GNU plot scripts.

   (b) Multi threading co-efficient:

   - In order to make relative comparison of tools, we select the tool which has the least multi-threading coefficient. We call this tool as the base tool.
   - We enumerate benchmarks as ERK= 1, LEVCHENKO= 2, ANGIOGENESIS= 3 and CIRCADIAN CLOCK= 4.
   - We perform normalization of the multi threading coefficient by diving the multi threading coefficient of each tool by the multi threading coefficient of the base tool.
   - On x axis we have the enumeration of the benchmarks and on y axis we have the normalized multi threading coefficient with respect to the base tool.
   - The normalized value shows that by how much factor the tool utilizes the multi threading property in comparison to other tools.

5.3 Benchmark ERK

5.3.1 Simulation in Snoopy

The average runtime and the peak memory consumption recorded for Snoopy for this model is given in Table 3 and Table 4 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0002	0.0018	0.1658	16.8030
	4	1.0003	1.0014	1.0079	4.9084
100	1	0.0009	0.0989	8.9830	884.4062
	4	1.0005	1.0084	3.0072	233.9180
10,000	1	0.0898	8.8005	875.7929	†
	4	1.0004	3.0085	231.8182	†
1,000,000	1	9.0178	889.899	†	†
	4	9.4008	243.3228	†	†

† runtime $> 3,600$ sec

Table 3: Snoopy, average runtime (in sec) for ERK.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	37,576	37,752	37,904	37,988
	4	37,512	37,767	37,824	39,604
100	1	37,580	39,748	39,507	39,968
	4	40,196	40,160	39,900	39,932
10,000	1	39,980	39,752	41,928	†
	4	40,032	40,112	39,568	†
1,000,000	1	39,600	39,460	†	†
	4	39,796	39,004	†	†

† runtime $> 3,600$ sec

Table 4: Snoopy, peak memory consumption (in KB) for ERK.

5.3.2 Simulation in StochKit

The conversion from SBML file to StochKit file took nearly 0.02 seconds and the peak memory consumption was 4,012 KB. The time was measured using the time command in Linux and the memory consumption was measured using a shell script.

The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 5 and Table 6 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0080	0.0134	0.4421	42.7381
	4	0.0080	0.0172	0.1284	11.5534
100	1	0.0080	0.0425	3.3970	338.0346
	4	0.0083	0.0276	0.9099	88.4614
10,000	1	0.0380	2.9934	294.4733	†
	4	0.0381	0.8301	77.7298	†
1,000,000	1	2.9665	296.6058	†	†
	4	2.9649	80.5857	†	†

† runtime $> 3,600$ sec

Table 5: StochKit, average runtime (in sec) for ERK.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	2,748	2,752	9,904	9,924
	4	2,748	2,748	20,560	24,800
100	1	2,748	2,748	9,982	9,924
	4	2,748	2,748	24,786	26,832
10,000	1	2,748	9,904	9,932	†
	4	2,748	24,800	24,876	†
1,000,000	1	9,964	9,968	†	†
	4	9,916	24,920	†	†

† runtime $> 3,600$ sec

Table 6: StochKit, peak memory consumption (in KB) for ERK.

5.3.3 Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 7 and Table 8 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0	0.0	0.0	13.0
	4	0.0	0.0	0.0	3.0
100	1	0.0	0.0	6.0	617.6
	4	0.0	0.0	1.0	162.5
10,000	1	0.0	6.0	620.3	†
	4	0.0	1.0	161.8	†
1,000,000	1	6.0	621.2	†	†
	4	6.0	163.5	†	†

† runtime $> 3,600$ sec

Table 7: Marcie average runtime (in sec) for ERK.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0,000	0,000	5,172	3,128
	4	3,126	0,000	5,324	5,324
100	1	0,000	5,172	3,132	3,156
	4	0,000	5,324	5,328	5,360
10,000	1	3,124	3,152	3,156	†
	4	3,128	5,328	5,356	†
1,000,000	1	3,148	3,160	†	†
	4	3,128	5,364	†	†

† runtime $> 3,600$ sec

Table 8: Marcie peak memory consumption (in KB) for ERK.

5.3.4 Simulation in Cain

The average runtime and the peak memory consumption recorded for CAIN for this model is given in Table 9 and Table 10 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0222	0.1160	8.1655	∗
	4	0.0231	0.0943	4.2285	∗
100	1	0.0231	0.1426	10.9276	∗
	4	0.0239	0.1048	5.1622	∗
10,000	1	0.0315	1.8656	183.8095	†
	4	0.0338	0.6837	60.1041	†
1,000,000	1	1.7569	173.7179	†	†
	4	1.7658	56.9456	†	†

† runtime $> 3,600$ sec
∗ tool crash while performing simulation

Table 9: CAIN average runtime (in sec) for ERK.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	110,048	113,544	201,795	∗
	4	110,520	117,912	109,125	∗
100	1	112,196	112,960	199,904	∗
	4	110,944	117,316	203,208	∗
10,000	1	110,080	114,324	199,728	†
	4	110,468	118,568	202,600	†
1,000,000	1	111,200	115,408	†	†
	4	111,792	118,192	†	†

† runtime $> 3,600$ sec
∗ tool crash while performing simulation

Table 10: CAIN peak memory consumption (in KB) for ERK.

5.3.5 Performance comparison

For runtime comparison of the tools refer Figure 9 which is plotted using Table 3 , Table 5, Table 7 and Table 9.

For memory comparison of the tools refer Figure 10 which is plotted using Table 4, Table 6, Table 8 and Table 10.

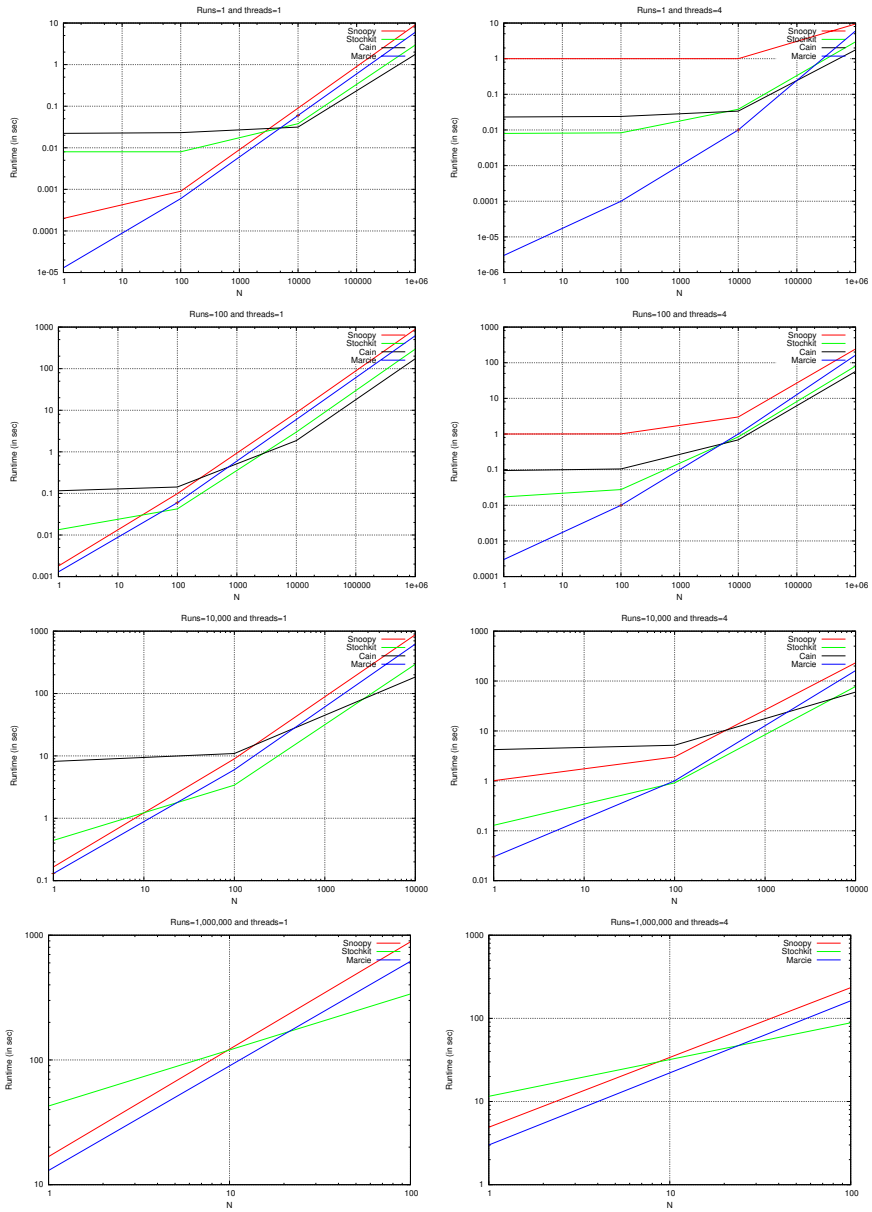

Figure 9: ERK, runtime comparison.

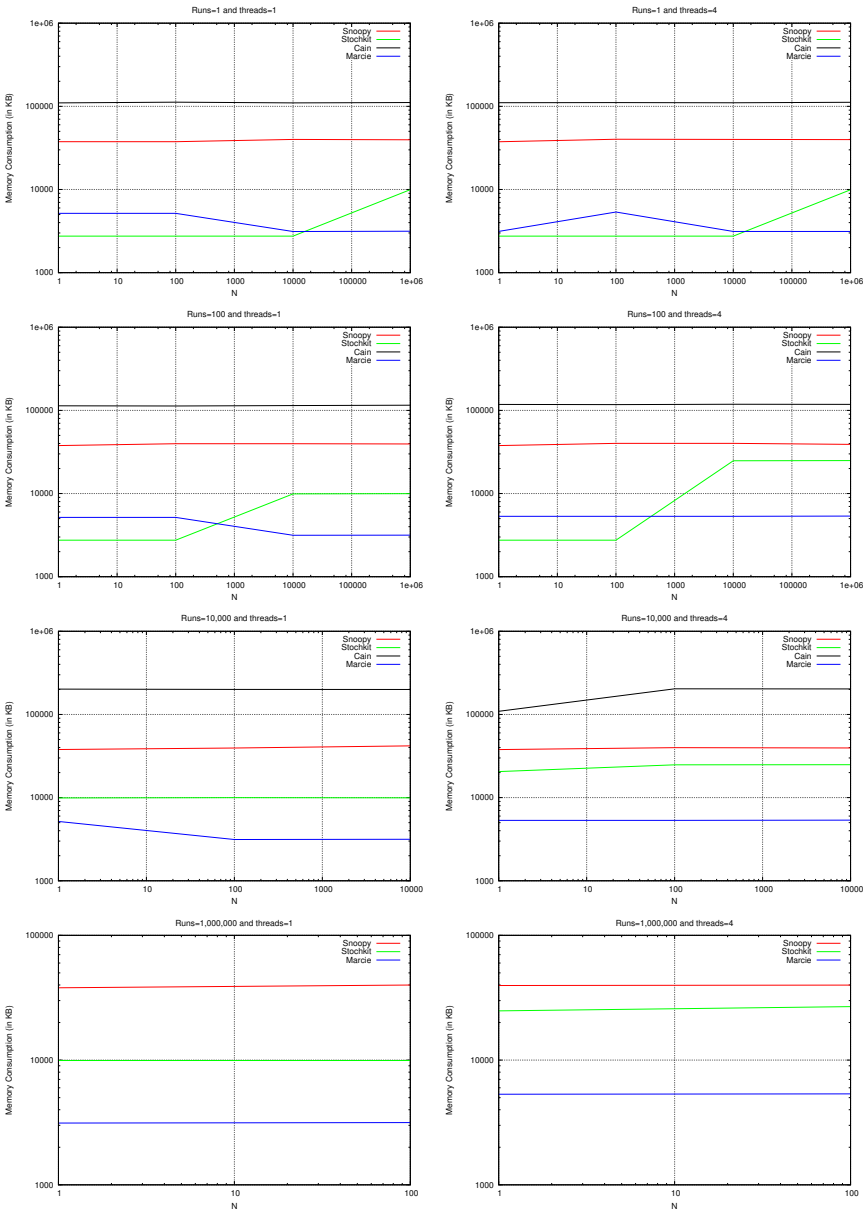

Figure 10: ERK, memory consumption comparison.

Tool	Threading Coefficient β	Variation
CAIN	3.0544	0.01
MARCIE	3.8112	0.02
SNOOPY	3.7387	0.07
STOCHKIT	3.7474	0.07

Table 11: Overall average threading coefficient of tools for ERK model

Because CAIN crashed for runs=1,000,000, it is not plotted in the graph. Also the values for which the simulation runtime is greater than 3,600 seconds are not plotted in the graph.
The peaks in the memory consumption plot in the figure 10 is due to the sampling rate of the script that we use. An explanation for this is already provided. Please refer section 5.1 *Note 2*.

5.4 Benchmark LEVCHENKO

5.4.1 Simulation in Snoopy

The average runtime and the peak memory consumption recorded for Snoopy for this model is given in Table 12 and Table 13 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0000	0.0105	1.0379	104.4715
	4	1.0002	1.0031	1.0070	27.9080
10	1	0.0011	0.1114	11.3115	1,066.5433
	4	1.0002	1.0086	3.4081	287.0238
100	1	0.0115	1.1429	112.4734	†
	4	1.0005	1.0084	30.6089	2,930.8798
1,000	1	0.1166	11.3270	1,152.1005	†
	4	1.0004	4.0074	295.7268	†

† runtime $> 3,600$ sec

Table 12: Snoopy, average runtime (in sec) for LEVCHENKO.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	41,804	43,140	45,752	45,636
	4	43,124	43,972	43,536	45,764
10	1	43,812	43,304	45,832	44,224
	4	41,772	45,744	43,840	47,064
100	1	41,528	43,132	43,528	†
	4	43,584	45,700	43,552	45,352
1,000	1	43,200	43,112	44,028	†
	4	43,412	43,508	43,952	†

† runtime $> 3,600$ sec

Table 13: Snoopy, peak memory consumption (in KB) for LEVCHENKO.

5.4.2 Simulation in StochKit

The conversion from SBML file to StochKit file took nearly 0.16 seconds and the peak memory consumption was 3,944 KB. The time was measured using the time command in Linux and the memory consumption was measured using a shell script.

The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 14 and Table 15 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0102	0.0215	1.0303	102.5429
	4	0.0103	0.0228	0.2912	26.3560
10	1	0.0108	0.0514	4.0852	406.9867
	4	0.0106	0.0296	1.0591	103.1096
100	1	0.0139	0.3583	33.8375	3,387.7400
	4	0.0139	0.1164	8.6536	852.9710
1,000	1	0.0460	3.3458	330.4806	†
	4	0.0449	0.9504	84.6891	†

† runtime $> 3,600$ sec

Table 14: StochKit, average runtime (in sec) for LEVCHENKO.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	3,108	3,116	3,144	3,116
	4	3,108	4,956	4,952	4,952
10	1	3,108	3,116	3,116	3,140
	4	3,108	4,956	4,952	4,952
100	1	3,108	3,116	3,136	3,140
	4	3,108	4,956	9,032	4,952
1,000	1	3,108	3,116	4,920	†
	4	3,108	4,952	4,952	†

† runtime $> 3,600$ sec

Table 15: StochKit, peak memory consumption (in KB) for LEVCHENKO.

5.4.3 Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 16 and Table 17, respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0	0.0	0.0	77.1
	4	0.0	0.0	0.0	20.0
10	1	0.0	0.0	7.0	731.6
	4	0.0	0.0	1.4	194.7
100	1	0.0	0.0	74.0	†
	4	0.0	0.0	20.0	1,997.2
1,000	1	0.0	7.0	757.2	†
	4	0.0	2.0	203.0	†

† runtime $> 3,600$ sec

Table 16: Marcie average runtime (in sec) for LEVCHENKO.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	3,408	3,408	3,408	3,408
	4	3,408	5,588	5,584	5,584
10	1	3,408	3,408	3,408	3,436
	4	3,408	5,584	5,576	5,612
100	1	3,408	3,408	3,432	†
	4	3,408	5,588	5,612	5,728
1,000	1	3,408	3,408	3,432	†
	4	3,408	5,588	5,616	†

† runtime $> 3,600$ sec

Table 17: Marcie peak memory consumption (in KB) for LEVCHENKO.

5.4.4 Simulation in CAIN

The average runtime and the peak memory consumption recorded for CAIN for this model is given in Table 18 and Table 19, respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0235	0.1632	12.8609	∗
	4	0.0255	0.1190	7.2922	∗
10	1	0.0243	0.1864	15.441	∗
	4	0.0270	0.1282	7.7260	∗
100	1	0.0244	0.3761	35.0678	∗
	4	0.0223	0.1715	11.7214	∗
1,000	1	0.0373	2.2283	220.4268	∗
	4	0.0371	0.6929	59.3757	∗

∗ tool crash while performing simulation

Table 18: CAIN average runtime (in sec) for LEVCHENKO.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	110,700	120,004	286,984	∗
	4	112,848	119,380	290,463	∗
10	1	111,308	119,832	289,760	∗
	4	111,780	122,208	292,713	∗
100	1	110,552	117,160	290,491	∗
	4	111,232	120,460	294,630	∗
1,000	1	112,764	119,044	288,492	∗
	4	113,432	119,742	294,838	∗

∗ tool crash while performing simulation

Table 19: CAIN peak memory consumption (in KB) for LEVCHENKO.

5.4.5 Performance comparison

For runtime comparison of the tools refer Figure 11 which is plotted using Table 12 , Table 14, Table 16 and Table 18.

For memory comparison of the tools refer Figure 12 which is plotted using Table 13, Table 15, Table 17 and Table 19.

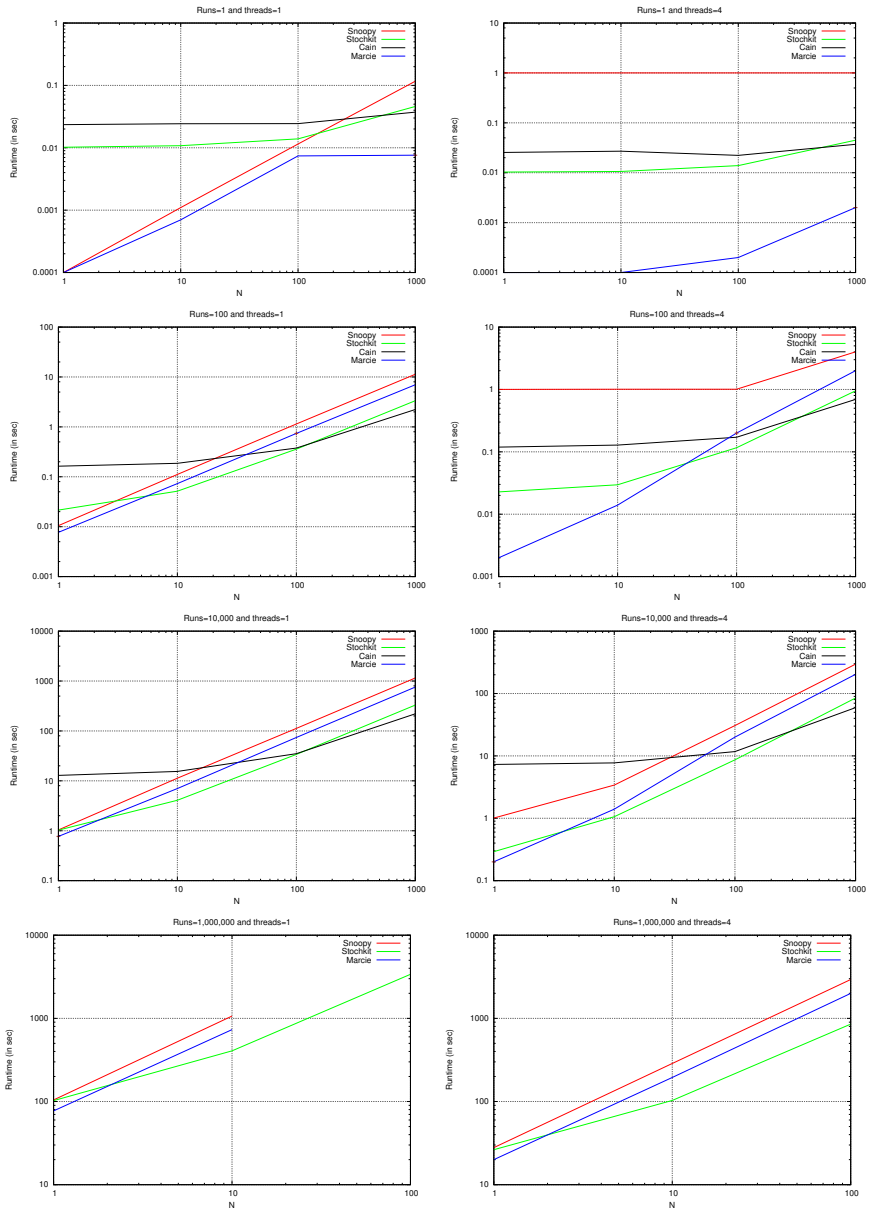

Figure 11: LEVCHENKO, runtime comparison.

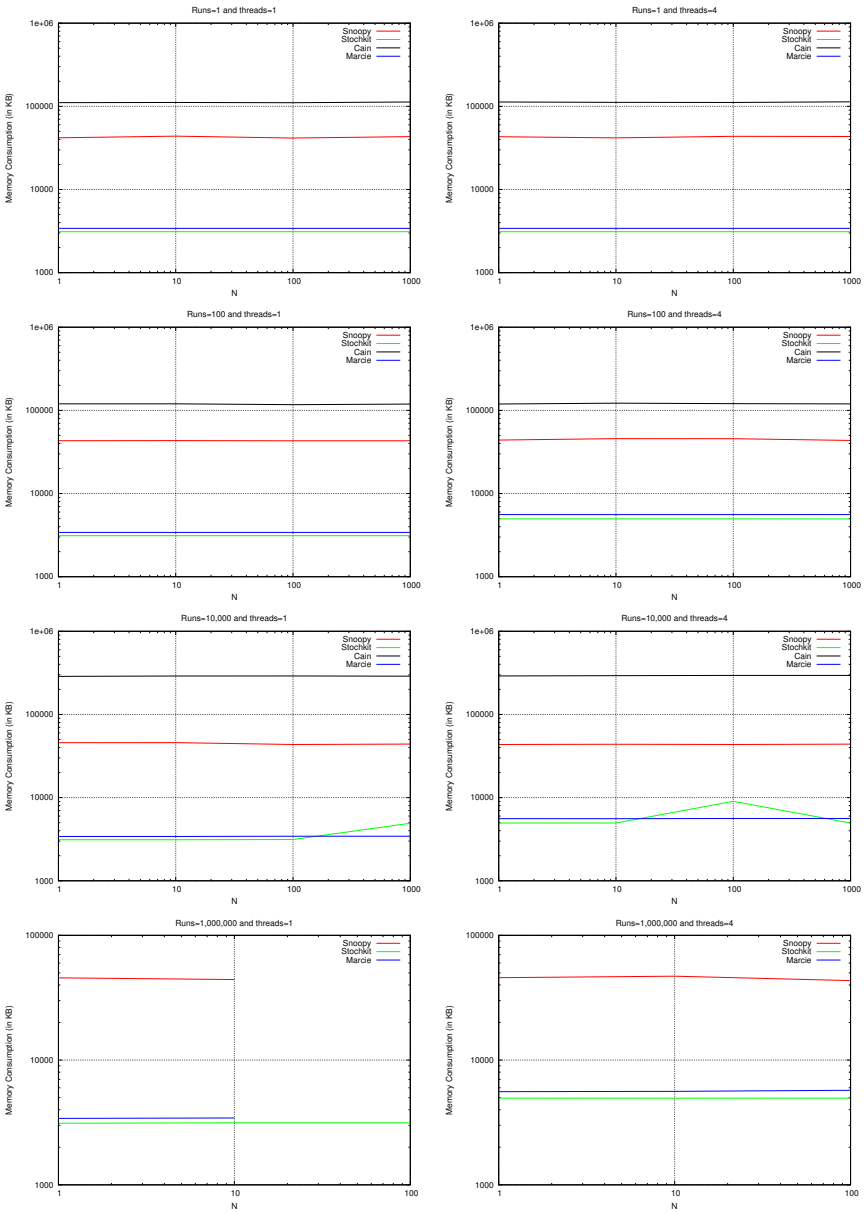

Figure 12: LEVCHENKO, memory consumption comparison.

Tool	Threading Coefficient β	Variation
CAIN	3.3521	0.51
MARCIE	3.7607	0.07
SNOOPY	3.7574	0.10
STOCHKIT	3.9280	0.04

Table 20: Overall average threading coefficient of tools for LEVCHENKO model

Since CAIN crashed for runs=1,000,000 , it is not plotted in the graph. Also the values for which the simulation runtime is greater than 3,600 seconds is also not plotted in the graph.
The peaks in the memory consumption plot in the figure 12 is due to the sampling rate of the script that we use. An explanation for this is already provided. Please refer section 5.1 *Note 2*.

5.5 Benchmark ANGIOGENESIS

5.5.1 Simulation in Snoopy

The average runtime and the peak memory consumption recorded for Snoopy for this model is given in Table 21 and Table 22 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0004	0.0317	3.2265	319.9296
	4	1.0001	1.0073	1.0058	84.3131
5	1	0.0035	0.3289	31.6073	3,169.8615
	4	1.0003	1.0060	9.0068	829.8543
10	1	0.0078	0.7567	74.1686	†
	4	1.0005	1.0060	20.7083	1,958.726
50	1	0.0476	4.6802	461.9907	†
	4	1.0003	2.0067	123.0167	†

† runtime $> 3,600$ sec

Table 21: Snoopy, average runtime (in sec) for ANGIOGENESIS.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	44,468	44,800	44,728	46,860
	4	45,008	45,194	44,832	45,120
5	1	44,772	45,132	44,784	45,264
	4	42,832	45,248	45,056	45,844
10	1	44,936	44,856	46,796	†
	4	44,976	45,268	45,260	45,492
50	1	44,704	44,732	45,600	†
	4	45,332	45,608	45,942	†

† runtime $> 3,600$ sec

Table 22: Snoopy, peak memory consumption (in KB) for ANGIOGENESIS.

5.5.2 Simulation in StochKit

The conversion from SBML file to StochKit file took nearly 0.14 seconds and the peak memory consumption was 3,952 KB. The time was measured using the time command in Linux and the memory consumption was measured using a shell script.

The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 23 and Table 24 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0138	0.0384	2.3848	235.6955
	4	0.0132	0.0335	0.6419	59.8950
5	1	0.0147	0.1605	14.4900	1,446.4240
	4	0.0147	0.0703	3.7421	367.4446
10	1	0.0161	0.3357	31.7338	3,178.9000
	4	0.0162	0.1055	8.0542	801.8241
50	1	0.0307	1.8246	179.7100	†
	4	0.0311	0.5007	45.0559	†

† runtime $> 3,600$ sec

Table 23: StochKit, average runtime (in sec) for ANGIOGENESIS.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	3,328	3,328	3,328	3,328
	4	3,328	5,276	5,276	5,276
5	1	3,328	5,240	3,328	3,328
	4	3,328	5,276	7,316	5,276
10	1	3,328	3,328	3,328	3,328
	4	3,328	5,276	5,276	5,400
50	1	3,328	3,328	3,328	†
	4	3,328	5,276	5,276	†

† runtime $> 3,600$ sec

Table 24: StochKit, peak memory consumption (in KB) for ANGIOGENESIS.

5.5.3 Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 25 and Table 26 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0	0.0	2.0	237.9
	4	0.0	0.0	0.0	59.8
5	1	0.0	0.0	23.0	2,300.3
	4	0.0	0.0	5.1	576.8
10	1	0.0	0.0	53.2	†
	4	0.0	0.0	13.0	1,344.8
50	1	0.0	3.0	313.3	†
	4	0.0	0.0	78.8	†

† runtime $> 3,600$ sec

Table 25: Marcie average runtime (in sec) for ANGIOGENESIS.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	3,628	3,632	3,628	3,632
	4	3,632	8,008	5,968	5,988
5	1	3,628	3,628	3,632	3,660
	4	3,632	5,968	5,960	6,000
10	1	3,632	3,632	3,652	†
	4	3,632	5,968	5,960	6,000
50	1	3,632	3,628	3,660	†
	4	3,628	5,964	5,996	†

† runtime $> 3,600$ sec

Table 26: Marcie peak memory consumption (in KB) for ANGIOGENESIS.

5.5.4 Simulation in CAIN

The average runtime and the peak memory consumption recorded for CAIN for this model is given in Table 27 and Table 28 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.0257	0.2595	22.2810	∗
	4	0.0266	0.1739	11.9135	∗
5	1	0.0239	0.3381	29.8249	∗
	4	0.0252	0.1915	12.2593	∗
10	1	0.0238	0.4484	41.0696	∗
	4	0.4490	0.2198	14.4687	∗
50	1	0.0251	1.3917	135.1852	∗
	4	0.0312	0.4808	38.5609	∗

∗ tool crash while performing simulation

Table 27: CAIN average runtime (in sec) for ANGIOGENESIS.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	111,936	120,640	426,497	∗
	4	112,664	123,944	424,200	∗
5	1	111,588	121,304	423,673	∗
	4	113,536	122,404	428,307	∗
10	1	112,384	127,992	423,659	∗
	4	112,228	122,816	426,925	∗
50	1	111,704	125,148	425,639	∗
	4	112,772	124,380	428,502	∗

∗ tool crash while performing simulation

Table 28: CAIN peak memory consumption (in KB) for ANGIOGENESIS.

5.5.5 Performance comparison

For runtime comparison of the tools refer Figure 13 which is plotted using Table 21 , Table 23, Table 25 and Table 27.

For memory comparison of the tools refer Figure 14 which is plotted using Table 22, Table 24, Table 26 and Table 28.

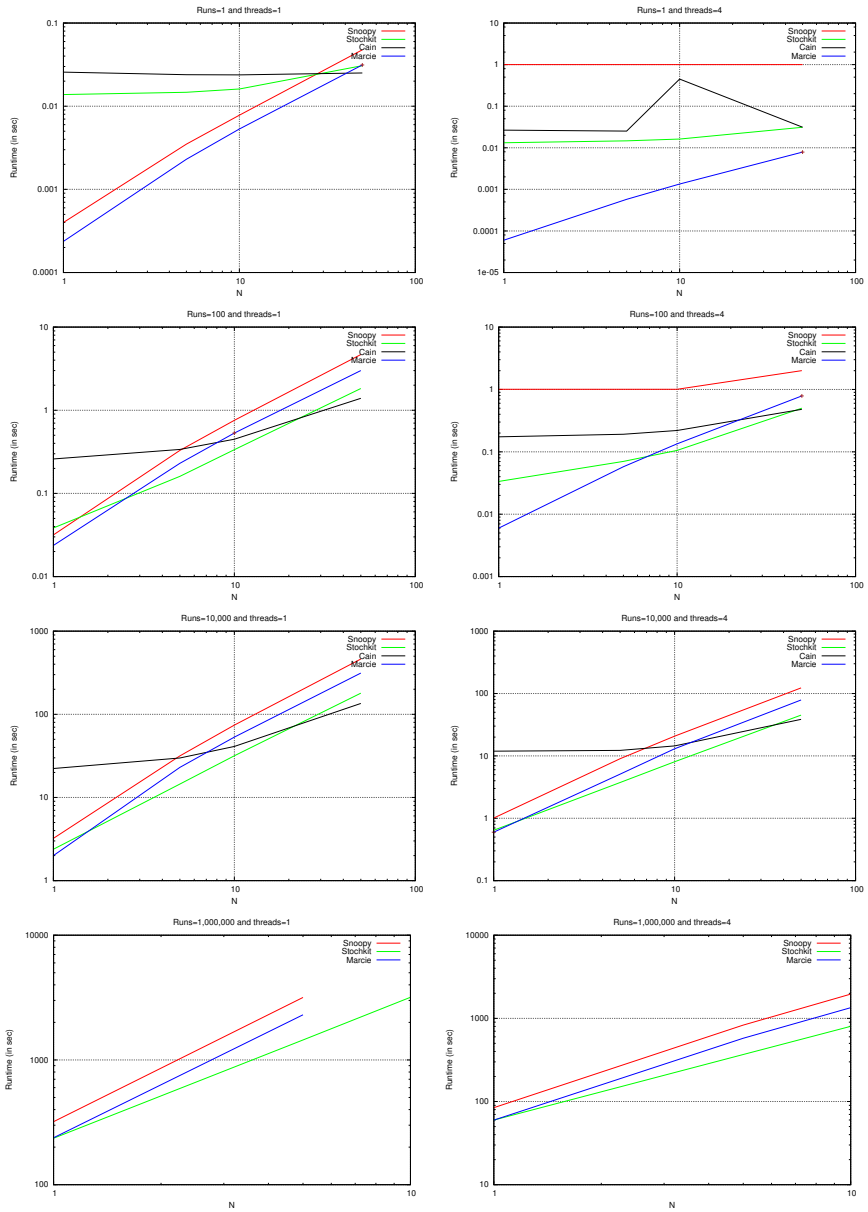

Figure 13: ANGIOGENESIS, runtime comparison.

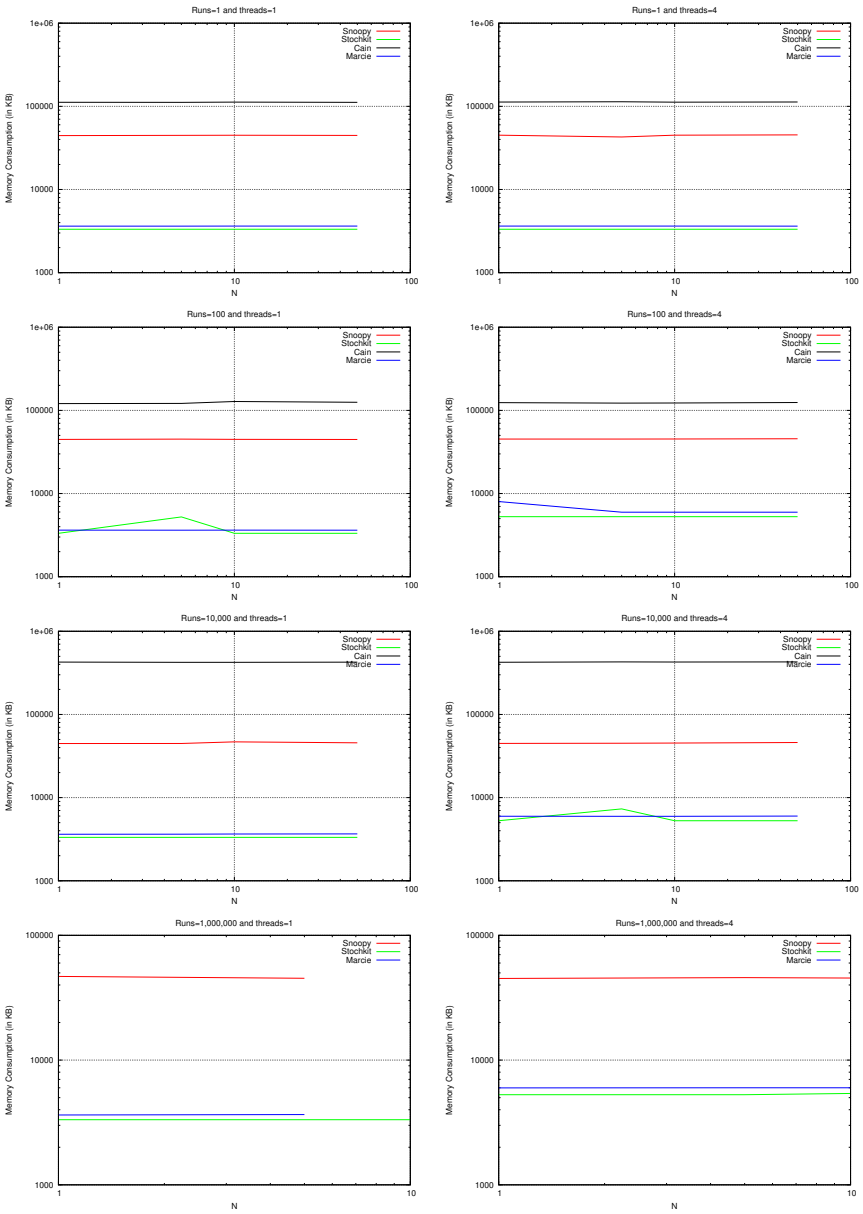

Figure 14: ANGIOGENESIS, memory consumption comparison.

Tool	Threading Coefficient β	Variation
CAIN	2.6757	0.71
MARCIE	3.9807	0.01
SNOOPY	3.7379	0.11
STOCHKIT	3.9562	0.03

Table 29: Overall average threading coefficient for ANGIOGENESIS model

Since CAIN crashed for runs=1,000,000 , it is not plotted in the graph. Also the values for which the simulation runtime is greater than 3,600 seconds is also not plotted in the graph.
The peaks in the memory consumption plot in the figure 14 is due to the sampling rate of the script that we use. An explanation for this is already provided. Please refer section 5.1 *Note 2*.

5.6 Benchmark CIRCADIAN CLOCK

5.6.1 Simulation in Snoopy

The average runtime and the peak memory consumption recorded for Snoopy for this model is given in Table 30 and Table 31 respectively.

	runs			
threads	1	100	10,000	1,000,000
1	0.1174	11.4623	1,140.9310	†
4	1.0003	3.9080	304.9245	†

† runtime $> 3,600$ sec

Table 30: Snoopy, average runtime (in sec) for CIRCADIAN CLOCK.

	runs			
threads	1	100	10,000	1,000,000
1	40,656	42,756	40,512	†
4	40,984	41,348	40,440	†

† runtime $> 3,600$ sec

Table 31: Snoopy, peak memory consumption (in KB) for CIRCADIAN CLOCK.

5.6.2 Simulation in StochKit

The conversion from SBML file to StochKit file took nearly 0.09 seconds and the peak memory consumption was 3,960 KB. The time was measured using the time command in Linux and the memory consumption was measured using a shell script. The average runtime and the peak memory consumption recorded for StochKit for this model is given in Table 32 and Table 33 respectively.

	runs			
threads	1	100	10,000	1,000,000
1	0.0606	5.2269	515.5103	†
4	0.0605	1.3854	130.9436	†

† runtime $> 3,600$ sec

Table 32: StochKit, average runtime (in sec) for CIRCADIAN CLOCK.

	runs			
threads	1	100	10,000	1,000,000
1	3,056	3,056	3,056	†
4	3,056	4,956	4,956	†

† runtime $> 3,600$ sec

Table 33: StochKit, peak memory consumption (in KB) for CIRCADIAN CLOCK.

5.6.3 Simulation in Marcie

The average runtime and the peak memory consumption recorded for Marcie for this model is given in Table 34 and Table 35 respectively.

	runs			
threads	1	100	10,000	1,000,000
1	0.0	10.6	866.3	†
4	0.0	4.0	217.1	†

† runtime $> 3,600$ sec

Table 34: Marcie, average runtime (in sec) for CIRCADIAN CLOCK.

	runs			
threads	1	100	10,000	1,000,000
1	3,236	3,264	3,264	†
4	3,236	3,396	5,344	†

† runtime $> 3,600$ sec

Table 35: Marcie, peak memory consumption (in KB) for CIRCADIAN CLOCK.

5.6.4 Simulation in CAIN

The average runtime and the peak memory consumption recorded for CAIN for this model is given in Table 36 and Table 37 respectively.

	runs			
threads	1	100	10,000	1,000,000
1	0.0450	2.6449	262.2521	†
4	0.0434	0.7796	68.9858	†

† runtime $> 3,600$ sec

Table 36: CAIN, average runtime (in sec) for CIRCADIAN CLOCK.

	runs			
threads	1	100	10,000	1,000,000
1	252,024	253,312	255,692	†
4	252,904	254,724	254,973	†

† runtime $> 3,600$ sec

Table 37: CAIN, peak memory consumption (in KB) for CIRCADIAN CLOCK.

5.6.5 Performance comparison

For runtime comparison of the tools refer Figure 15 which is plotted using Table 30 , Table 32, Table 34 and Table 36.
For memory comparison of the tools refer Figure 16 which is plotted using Table 31, Table 33, Table 35 and Table 37.

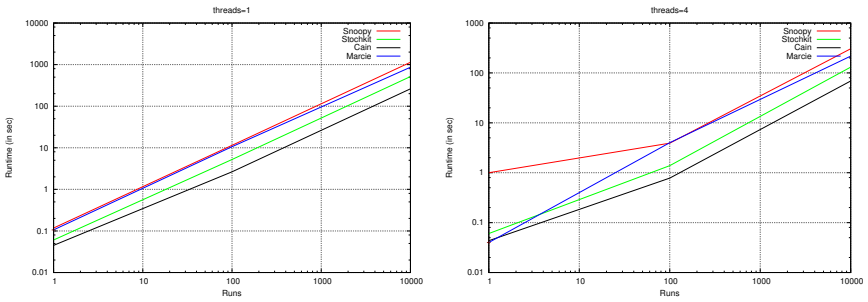

Figure 15: CIRCADIAN CLOCK, runtime comparison.

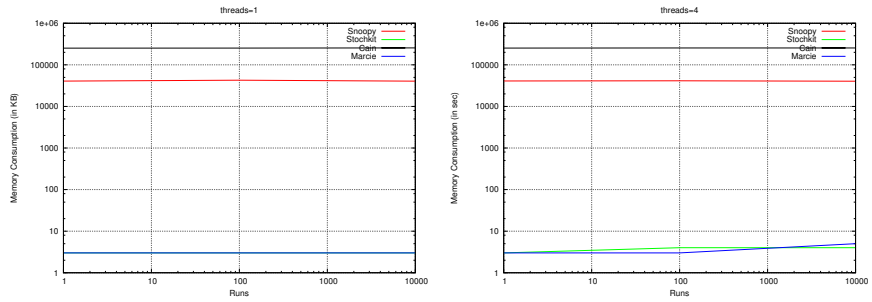

Figure 16: CIRCADIAN CLOCK, memory consumption comparison.

Tool	Threading Coefficient β	Variation
CAIN	3.8015	$\wedge$
MARCIE	3.9903	$\wedge$
SNOOPY	3.7417	$\wedge$
STOCHKIT	3.9369	$\wedge$

$\wedge$ only one sample reading

Table 38: Overall average threading coefficient of tools for CIRCADIAN CLOCK model

The values for which the simulation runtime is greater than 3,600 seconds is also not plotted in the graph. The variance column in Table 38 contains $\wedge$ because the readings has only one sample point. Variance cannot be calculated if there is only 1 sample point.
The peaks in the memory consumption plot in the figure 16 is due to the sampling rate of the script that we use. An explanation for this is already provided. Please refer section 5.1 *Note 2*.

5.7 Observations

Some of the observations while performing simulations were:

- While performing simulation on CAIN for ERK benchmark, the memory consumption was nearly 726,320 KB when the simulation was performed continuously for N=1, runs=10,000, thread=1, without closing CAIN. This is about 4 times the peak memory consumption when opening and closing the tool for each single experiment. A memory leak is the most probable explanation.
- While performing simulation on CAIN for numbers of runs= 1,000,000, most of the time we encountered a crash. Memory leak is the probable explanation for this crash i.e. CAIN runs out of memory while performing simulation.
- The CSV export in CAIN takes some time for large number of runs ($\geq$10,000). The export time is more if the number of species in the model is more. For ERK model it took almost 90 seconds for its export for runs= 10,000.
- In StochKit, the 'ssa' driver uses information about the model to try to select the simulation method that will achieve the best performance.

5.8 Comparison

5.8.1 Accuracy

We calculate relative accuracy in this case. For all species in a reaction, we plot the traces generated by all the tools. For every specie in the reaction, for increasing value of scaling parameter and runs, the plot should converge. For a particular value scaling parameter and no of runs, the plots will differ slightly for different values of threads. This is due to stochasticity.
The results obtained were similar to the expected result and we can say that the simulation performed was correct.
For more details please refer appendix.

5.8.2 Simulation runtime comparison

The tools are compared on the basis of their average simulation runtime for a particular benchmark for a particular value of run. The particular value of run is determined by the maximum value of run on which all the tools have simulation time $<$ 3,600 sec and tool doesn't crashes. In this case we have selected the run value as 10,000.
The relative run time of tools is plotted. We term this as relative runtime because we plot the difference between the runtime of the tools. We select the tool which has the maximum runtime. We call this tool as the base tool.

The runtime of each tool is subtracted from the runtime of the slowest tool. The positive value shows by how much value the tool is faster than the base tool. The negative value show by how much value the tool is slower than the base tool. On x-axis we have the scaling parameter and on y-axis we have the relative runtime in sec.

N	threads	Tools			
		Snoopy	StochKit	Marcie	CAIN
1	1	0.1658	0.4421	0.1300	8.1655
	4	1.0079	0.1284	0.0300	4.2285
100	1	8.9830	3.3970	6.0000	10.9276
	4	3.0072	0.9099	1.0000	5.1622
10,000	1	875.7929	294.4733	620.3000	183.8095
	4	231.8182	77.7298	161.8000	60.1041
1,000,000	1	†	†	†	†
	4	†	†	†	†

† runtime $> 3,600$ sec

Table 39: Overall average runtime (in sec) for ERK.

N	threads	Tools			
		Snoopy	StochKit	Marcie	CAIN
1	1	1.0379	1.0303	0.7710	12.8609
	4	1.0070	0.2912	0.2000	7.2922
10	1	11.3115	4.0852	7.0000	15.4410
	4	3.4081	1.0591	1.4000	7.7260
100	1	112.4734	33.8375	74.0000	35.0678
	4	30.6089	8.6536	20.0000	11.7214
1,000	1	1,152.1005	330.4806	757.2000	220.4268
	4	295.7268	84.6891	203.0000	59.3757

Table 40: Overall average runtime (in sec) for LEVCHENKO.

N	threads	Tools			
		Snoopy	StochKit	Marcie	CAIN
1	1	3.2265	2.3848	2.0000	22.2810
	4	1.0058	0.6419	0.5980	11.9135
5	1	31.6073	14.4900	23.0000	29.8249
	4	9.0068	3.7421	5.1000	12.2593
10	1	74.1686	31.7338	53.2000	41.0696
	4	20.7083	8.0454	13.0000	14.4687
50	1	461.9907	179.7100	313.3000	135.1852
	4	123.0167	45.0559	78.8000	38.5609

Table 41: Overall average runtime (in sec) for ANGIOGENESIS.

Runs	threads	Tools			
		Snoopy	StochKit	Marcie	CAIN
1	1	0.1174	0.0606	0.1060	0.0450
	4	1.0030	0.0605	0.0400	0.4340
100	1	11.4623	5.2269	10.6000	2.6449
	4	3.9080	1.3854	4.0000	0.7796
10,000	1	1140.9310	515.5103	866.3000	262.2521
	4	304.9245	130.9436	217.1000	68.9858
1,000,000	1	†	†	†	†
	4	†	†	†	†

† runtime $> 3,600$ sec

Table 42: Overall average runtime (in sec) for CIRCADIAN CLOCK.

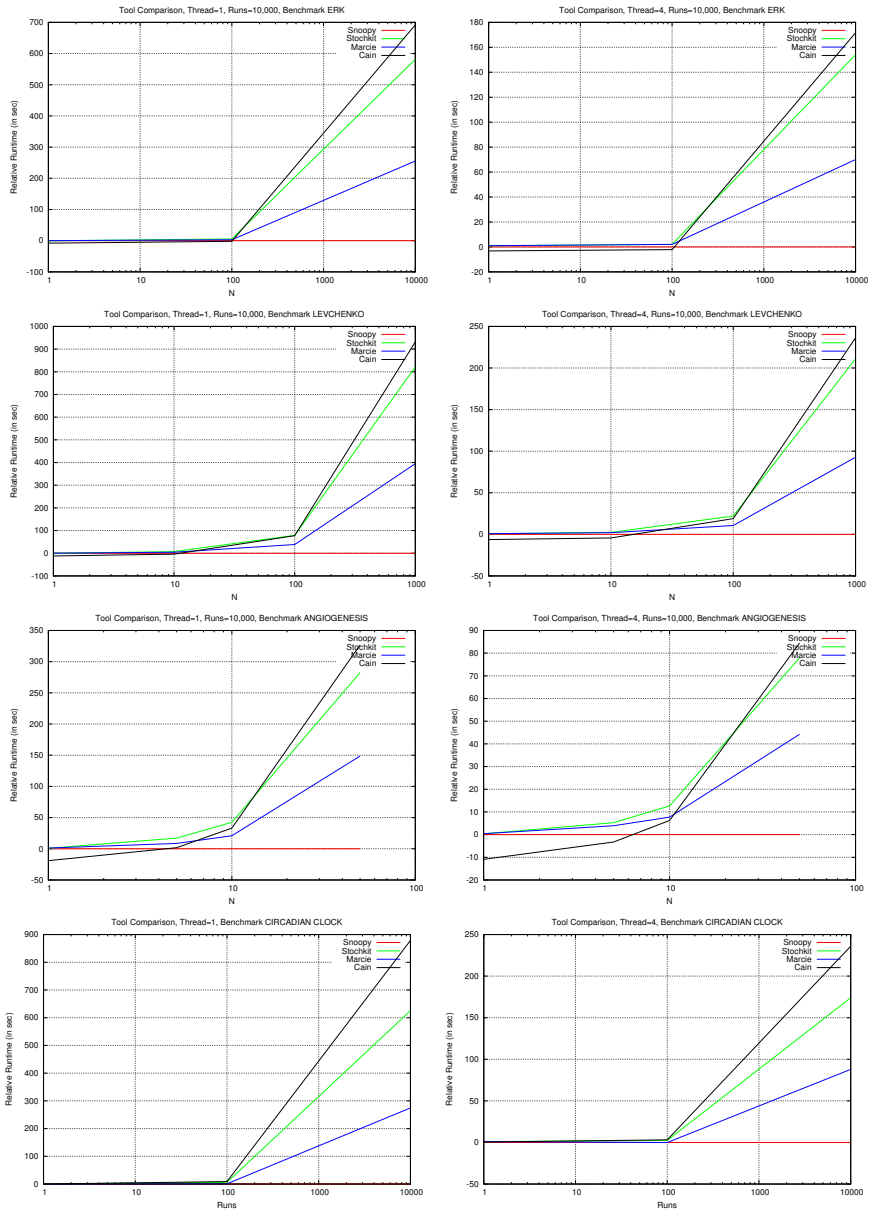

Figure 17: Overall relative runtime comparison of tools

Interpretation of the plot: For the ERK benchmark, for threads=1 and Runs=10,000, top left in the figure 17, we have snoopy as the base tool because it has the maximum runtime. For N $\leq$ 100 the Marcie, Snoopy and StochKit graph overlap each other. This signifies that there is not much difference between the runtime of the tools if it is performed for N $\leq$ 100. However CAIN lies below Snoopy which means that for smaller value of N, Snoopy is faster than CAIN. However for N $>$ 100 is above StochKit which is above Marcie which is above Snoopy. This means that CAIN is the fastest among all the tools for a specified number of runs (here runs= 10,000).For N= 1,000,000 CAIN takes nearly 700 seconds less than Snoopy, StochKit takes nearly 580 seconds less than Snoopy, Marcie takes nearly 250 seconds less than Snoopy while performing simulation for runs= 10,000.
Conclusions:

1. CAIN comes to be fastest of all the tools. CAIN crashes most of the times for runs= 1,000,000. So CAIN does not seems to be a reliable tool for large number of runs.

2. Stochkit is a fast tool in comparison to other tools and it is a reliable for runs up to 1,000,000.

    - In case of levchenko benchmark it was the only tool which survived for N= 100, threads= 1 and runs= 1,000,000 refer table 14 and figure 11. Snoopy and Marcie had simulation runtime $>$ 3,600 sec refer table 12 and table 16. CAIN crashed while performing simulation for runs= 1,000,000 on levchenko benchmark refer table 18.
    - In case of angiogenesis benchmark it was the only tool which survived for N= 10, threads= 1 and runs= 1,000,000 refer table 23 and figure 13. Snoopy and Marcie had simulation runtime $>$ 3,600 sec refer table 21 and table 25. CAIN crashed while performing simulation for runs= 1,000,000 on angiogenesis benchmark refer table 27.

    So, we treat StochKit as a fast and reliable tool.

3. Marcie is faster than Snoopy but slower than StochKit in terms of runtime but it has a special feature of model checking. Marcie can also be treated as a reliable tool since it does not crashes for runs up to 1,000,000.

4. Snoopy was found to be slowest of all the tools, but it is an extremely easy GUI tool in handling. Snoopy can also be treated as a reliable tool since it does not crashes for runs up to 1,000,000.

Ranking of the tools on the basis of runtime:

1. CAIN
2. StochKit
3. Marcie
4. Snoopy

Ranking of the tools on the basis of reliability:

1. StochKit
2. Marcie $\simeq$ Snoopy
3. CAIN

5.8.3 Memory consumption comparison

The memory consumption plots are plotted for runs=10,000 and for each benchmark. The graphs are directly taken from the memory comparison section of each benchmark for runs= 10,000. The An explanation for the peak is already provided. Please refer section 5.1 *Note 2*.
The plots in the first, second, third and fourth row are for ERK, LEVCHENKO, ANGIOGENESIS, CIRCADIAN CLOCK benchmarks respectively. Plots in the left hand side are for threads= 1 while on the right hand side are for threads= 4.
Conclusions:

1. The command line tools will use less memory than the GUI tools for the same benchmark and parameter value. So the comparison will be based on the interface type i.e. GUI and command line.
2. Marcie is the most efficient tool in terms of memory consumption.
3. CAIN has some issues with its memory consumption as it crashes for runs=1,000,000.
4. StochKit uses slightly higher memory than Marcie.

Ranking of the tools on the basis of memory:

- GUI tool:
  1. Snoopy
  2. CAIN
- Command line tool:
  1. Marcie
  2. StochKit

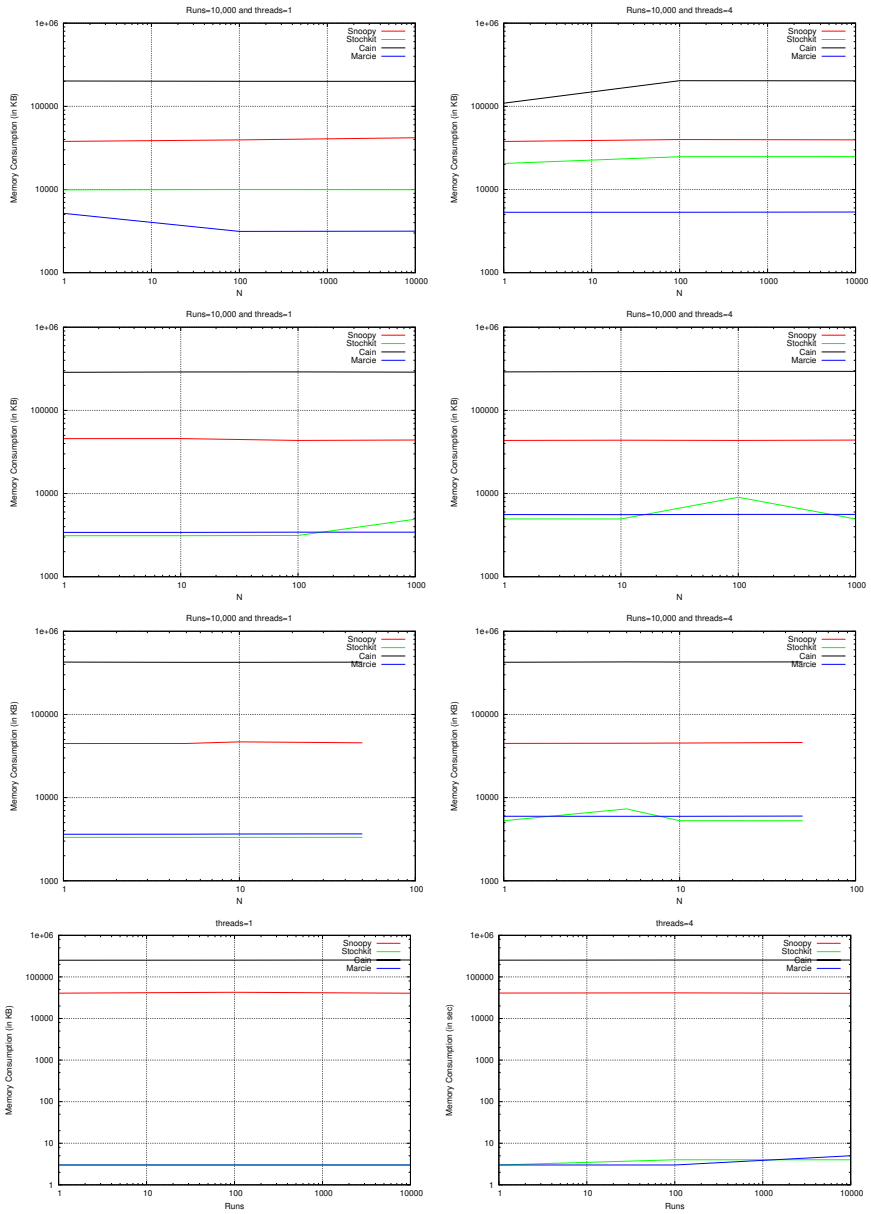

Figure 18: Overall memory consumption of tools

5.8.4 Multi-threading coefficient (β)

The benchmarks are enumerated in order to plot this value. The enumeration value for ERK is 1, LEVCHENKO is 2, ANGIOGENESIS is 3 and CIRCADIAN CLOCK is 4. We select a base tool which has the minimum value of multi-threading coefficient (β). In our case the base tool is CAIN.

	Tools			
Benchmarks	Snoopy	StochKit	CAIN	Marcie
1	1.2240	1.2269	1.0000	1.2478
2	1.1209	1.1718	1.0000	1.1219
3	1.3970	1.4786	1.0000	1.4786
4	0.9843	1.0356	1.0000	1.0497

Table 43: Multi-threading coefficient comparison

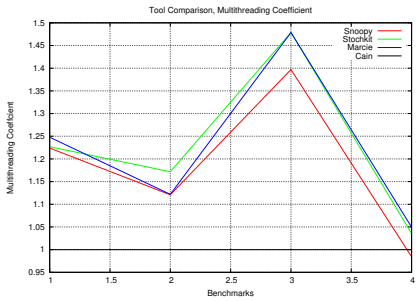

Figure 19: Relative Multi-threading coefficient of tools

Interpretation of the plot: The x-axis denote the enumerated benchmarks, however the values such as 1.5, 2.5 and 3.5 doesn't signify anything. It has been generated by the GNU plot script in order to visualize the plot clearly. CAIN is the base tool and the multi-threading value of CAIN is kept as 1. For benchmark 1 i.e. ERK Marcie utilizes the multi-cores of the system nearly about 1.25 time better than CAIN. Similarly, StochKit utilizes multi-cores of the system better than Marcie which utilizes the multi-cores of the system better than CAIN. If the lines are below the line of the base tool then the base tool utilizes the multi-cores of the system more efficiently. For examples, for the benchmark 4, i.e. CIRCADIAN CLOCK Snoopy has less multi-threading coefficient than CAIN.

Conclusion

Ranking of the tools on the basis of multi-threading coefficient:

1. StochKit
2. Marcie
3. Snoopy
4. CAIN

6 Summary

After performing simulation on different tools and different benchmarks for different parameter values, the selected tools were compared. The comparison was based on qualitative as well as quantitative criteria.
For a GUI tool I would prefer Snoopy over CAIN. The reasons are:

- The main reason is reliability the tool. Snoopy is more reliable than CAIN for runs> 10,000. This is because CAIN crashes for runs= 1,000,000.
- Snoopy consumes lesser memory than CAIN.
- Snoopy has a better multi-threading coefficient value than CAIN.
- CAIN takes longer time to export traces for number of species in a reaction > 10 for runs= 10,000.
- Snoopy is an extremely easy tool to handle.

For a command line tool I would prefer Stochkit over Marcie. the reasons are:

- The main reason is reliability the tool. StochKit performs simulation of few benchmarks within 3,600 sec while Marcie takes more than 3,600 sec for the same parameter values. Refer figure 11
- Although Marcie is most efficient tool in terms of memory consumption, StochKit uses slightly higher memory than Marcie which can be accepted.
- StochKit and Marcie are nearly equivalent in terms of multi-threading coefficient value.
- StochKit has a special property. Its 'ssa' driver uses information about the model and tries to select the simulation method that will achieve the best performance. Whereas Marcie has a special feature of model compilation.

6.1 Achievements

We have developed an evaluation protocol for comparing different tools. The protocol consists of qualitative and quantitative criteria for comparing different tools. Based on these criteria we determine the performance of the tool. This evaluation can be further extended to incorporate different tools. Scripts were developed in C, C++, C# and some shell scripts. The report is written in latex and I learned some new technologies like shell scripts and latex.

6.2 Open Problems

There are few potential areas where this work can be extended. Few of them are:

1. The evaluation protocol can be extended to benchmarks which are scalable in size. We could not do so because of our time constraints.

2. This work can be extended for other tools which do not support SBML.

3. There are few potential candidates which need to be analysed and can be incorporated as the part of comparison study. BioNetGen, V-Cell and model compilers such as SSC can be potential candidates.

4. We have limited ourselves to the tools which have C/C++ as their implementation language. However, we can compare tools which are implemented in other languages such as Java, Python etc.

References

[1] F. Cordero, A. Horváth, D. Manini, L. Napione, M. D. Pierro, S. Pavan, A. Picco, A. Veglio, M. Sereno, F. Bussolino, and G. Balbo. Simplification of a complex signal transduction model using invariants and flow equivalent servers. *Theor. Comput. Sci.*, 412(43):6036–6057, 2011.

[2] D. Gilbert and M. Heiner. From Petri nets to differential equations - an integrative approach for biochemical network analysis. In *Proc. ICATPN 2006*, pages 181–200. LNCS 4024, Springer, 2006.

[3] D. Gilbert, M. Heiner, and S. Lehrack. A unifying framework for modelling and analysing biochemical pathways using Petri nets. In *Proc. CMSB*, pages 200–216. LNCS/LNBI 4695, Springer, 2007.

[4] M. Heiner, R. Donaldson, and D. Gilbert. *Petri Nets for Systems Biology*, chapter 3, pages 61–97. Jones & Bartlett Learning, LCC, 2010.

[5] M. Heiner, D. Gilbert, and R. Donaldson. *Petri Nets for Systems and Synthetic Biology*, volume 5016 of *LNCS*, pages 215–264. Springer, 2008.

[6] M. Heiner, R. Richter, and M. Schwarick. Snoopy - a tool to design and animate/simulate graph-based formalisms. In *Proc. International Workshop on Petri Nets Tools and APplications (PNTAP 2008, associated to SIMUTools 2008)*. ACM digital library, 2008.

[7] M. Herajy and M. Heiner. Snoopy Computational Steering Framework User Manual Version 1.0. Technical Report 02-13, Brandenburg University of Technology Cottbus, Department of Computer Science, July 2013.

[8] K. hyun Cho, S. young Shin, H. woo Kim, O. Wolkenhauer, B. Mcferran, and W. Kolch. Mathematical modeling of the influence of rkip on the erk signaling pathway. In *In C. Priami, editor, Computational Methods in Systems Biology (CSMB03), volume 2602 of LNCS*, pages 127–141. Springer-Verlag, 2003.

[9] A. Levchenko, J. Bruck, and P. Sternberg. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *Proc. Natl. Acad. Sci. USA*, 97(11):5818–23, 2000.

[10] F. Liu and M. Heiner. *Petri Nets for Modeling and Analyzing Biochemical Reaction Networks*, chapter 9, pages 245–272. Springer, 2014.

[11] W. Marwan, C. Rohr, and M. Heiner. *Petri nets in Snoopy: A unifying framework for the graphical display, computational modelling, and simulation of bacterial regulatory networks*, volume 804 of *Methods in Molecular Biology*, chapter 21, pages 409–437. Humana Press, 2012.

[12] S. L. Naama Barkai. Biological rhythms: Circadian clocks limited by noise, 2000.

[13] L. Napione, D. Manini, F. Cordero, A. Horváth, A. Picco, M. D. Pierro, S. Pavan, M. Sereno, A. Veglio, F. Bussolino, and G. Balbo. On the use of stochastic petri nets in the analysis of signal transduction pathways for angiogenesis process. In P. Degano and R. Gorrieri, editors, *CMSB*, volume 5688 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2009.

[14] C. Rohr. Simulative csl model checking of stochastic petri nets in iddmc. In *Proc. 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010)*, volume 643 of *CEUR Workshop Proceedings*, pages 88–93. CEUR-WS.org, October 2010.

[15] K. R. Sanft, S. Wu, M. K. Roh, J. Fu, R. K. Lim, and L. R. Petzold. Stochkit2: software for discrete stochastic simulation of biochemical systems with events. *Bioinformatics*, 27(17):2457–2458, 2011.

[16] M. Schwarick. *Manual: Marcie - An analysis tool for Generalized Stochastic Petri nets*. BTU Cottbus, Dep. of CS, 2011.

[17] M. Schwarick and M. Heiner. CSL model checking of biochemical networks with interval decision diagrams. In *Proc. 7th International Conference on Computational Methods in Systems Biology (CMSB 2009)*, volume 5688 of *LNCS/LNBI*, pages 296–312. Springer, September 2009.

[18] J. Vilar, H.-Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proc. National Academy of Sciences of the United States of America*, 99(9):5988–5992, 2002.

[19] D. Wilkinson. Gibbs sampler in various languages (revisited). 2011.

Appendices

A Accuracy

A.1 Accuracy of a stochastic simulation tool

In order to perform experiment we must be sure about the correctness of the experiment. We have incorporated four tools in our comparison study and we would like to check whether the tools are performing simulations correctly or not. There will be differences in the graph produced by plotting the traces obtained from the tools due to stochasticity. However if we do averaging of traces for higher number of runs we may check how accurate is the tool.

A.2 How to determine accuracy

Accuracy in this case is defined relatively. The word relatively means that we don't have any exact calculations or benchmarks on which we can test the accuracy. However we developed an evaluation protocol to determine the relative accuracy of the tool.
On changing the thread value keeping the number of runs and the value of scaling parameter constant for a particular benchmark model, we should not get much difference between the curves. There will be slight difference in the curve due to stochasticity.
On increasing the runs keeping the scaling parameter constant and number of threads constant, we should expect that the curves of a specie should become smoother and should converge with the curves produced by other tools.
On increasing the runs keeping the scaling parameter constant and number of threads constant, we should expect that the curves of a particular specie should become smooth and should converge with the curves produced by other tools.
We will plot these curves for some species only from each benchmark.

A.3 Technology

The scripts run for the benchmarks having scaling parameter. There are separate scripts for benchmarks having no scaling parameter, e.g. Circadian Clock. Scripts for benchmarks with no scaling parameter will have the benchmark name attached to script name, e.g. standard_file_cir means.cpp means it is script for renaming the files to the standard naming convention for Circadian Clock benchmark. The following steps were followed in order to obtain the accuracy curve:

1. For each experiment we should have 10 trials and we should store the traces of each trial.

2. The traces from each tool should follow the standard file and folder naming convention as explained in section 5.2.

3. Traces which are exported from Snoopy, CAIN and Marcie has ".csv" file extension with "," as the datafile separator. Traces which are exported from StochKit, ".txt" file extension with "tab" as the datafile separator.

4. There is a C++ script "standard names.cpp" which needs to be run. Run the script only for tools CAIN, Snoopy and Marcie. For StochKit there is a separate script "standard_names_stochkit.cpp". This is because StochKit saves the traces in a folder and the StochKit trace file name is "means.txt". If the naming convention is not followed, we can still use the script and modify the script providing the naming convention which the user followed while performing the experiments. The script renames the file to the standard naming convention and changes the extension of the trace file to ".txt". However the datafile separator of these files are not changed. The standard names ensures that all the trace files are in standard naming format and have ".txt" as its extension. The purpose of changing extension is to bring all the traces in ".txt" format which can be used to plot from GNU plot scripts. However one can also use ".csv" extension or extension of his/her choice as GNU plot supports various extensions. For simplicity I have used ".txt" as the file extension. For more details about GNU plot please refer `http://people.duke.edu/~hpgavin/gnuplot.html`.

5. After running the scripts for all the tools, run the script "copying_file.cpp". You have the choice to decide the path of the folder where you would like to copy the files. This will copy a file to a standard location.

6. The location which you specified in the previous script "copying_files.cpp" has at most 32 trace files for each tool. As mentioned earlier the datafile separator for StochKit trace file is "tab", we open the trace files and using the replace function of the editor, we just replace the "tab" with "," and save the files.

7. We will now generate GNU plot scripts using a C++ script. The script "gnuplotscript.cpp" should be edited prior to its execution. We need to specify the value of the scaling parameter for the benchmark, the species in the reactions in the same order as in the trace file and our desired path. We now run "gnuplotscripts.cpp". The script generates 32 files for each experiment.

8. There are some terminal commands written in "folder_commands_accuracy.txt". We have to just specify the name of the species in the file. These commands generate a folder "plot_experiment". "plot_experiment" has sub-folders and each folder denotes a specie folder. Make sure that the final traces file (32 files for each tool), GNU plot script (generated by the "gnuplotscripts.cpp") and the folder "plot_experiments" are at same location.

9. Plot the GNU plot script files. Curves for each specie is automatically generated and can be found in the specie sub-folder.

A.4 Results

Expectations:

1. The curves should converge on increasing the number of runs keeping the scaling parameter constant and thread constant.

2. The curves should converge on increasing the value of scaling parameter keeping the runs constant and thread constant.

3. The curves should look similar when plotted for different threads keeping runs constant and scaling parameter constant.

Here we take one specie from each benchmark and plot the curves. For ERK benchmark we plot the specie MEKPP. For Levchenko benchmark we plot the specie ERK_MEKPP. For Angiogenesis benchmark we plot the specie GStarP3. For Circadian Clock benchmark we plot the specie a.

A.4.1 Variation with the values of scaling parameter

We are not plotting curves for Circadian Clock benchmark here because it has no scaling parameter. Here runs are kept constant. Value of runs= 100.

ERK benchmark, specie: MEKPP

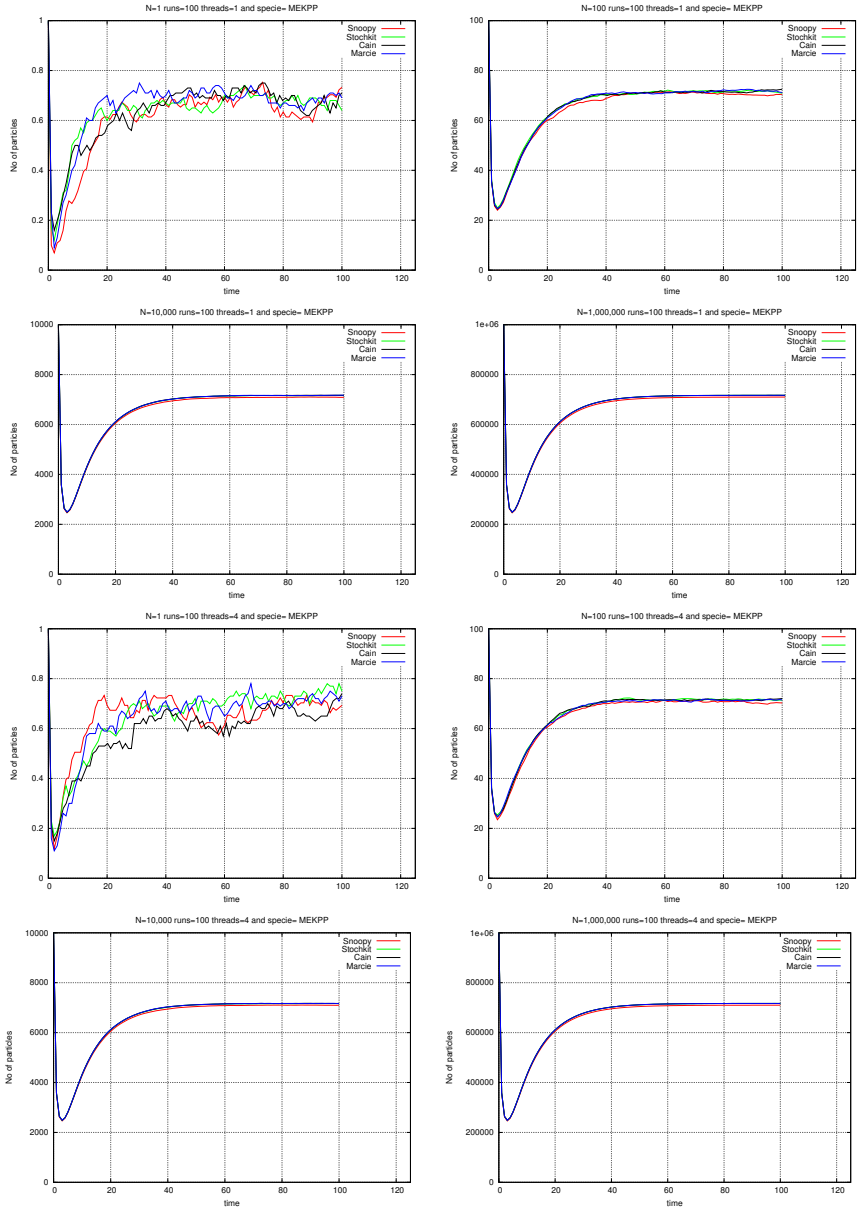

Figure 20: ERK, accuracy with varying scaling parameter.

LEVCHENKO benchmark, specie: ERK_MEKPP

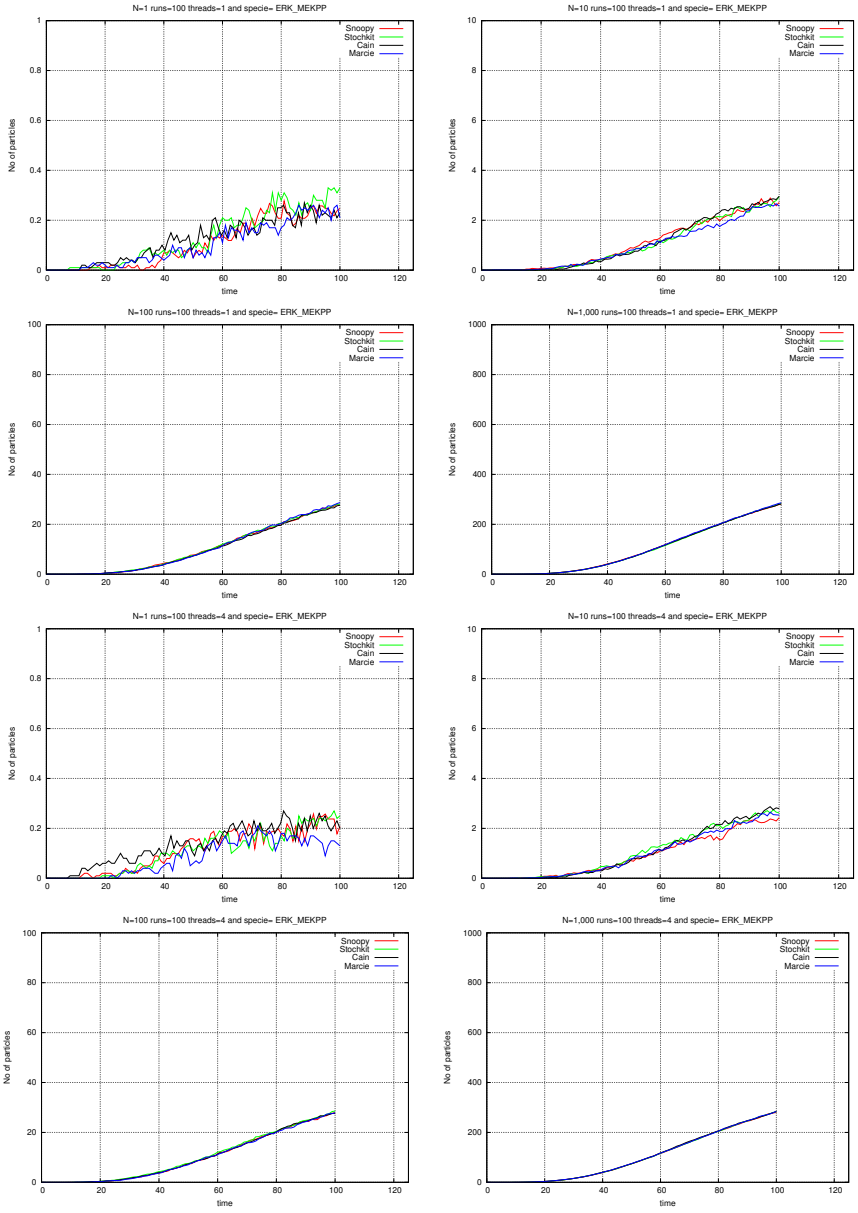

Figure 21: LEVCHENKO, accuracy with varying scaling parameter.

ANGIOGENESIS benchmark, specie: GStarP3

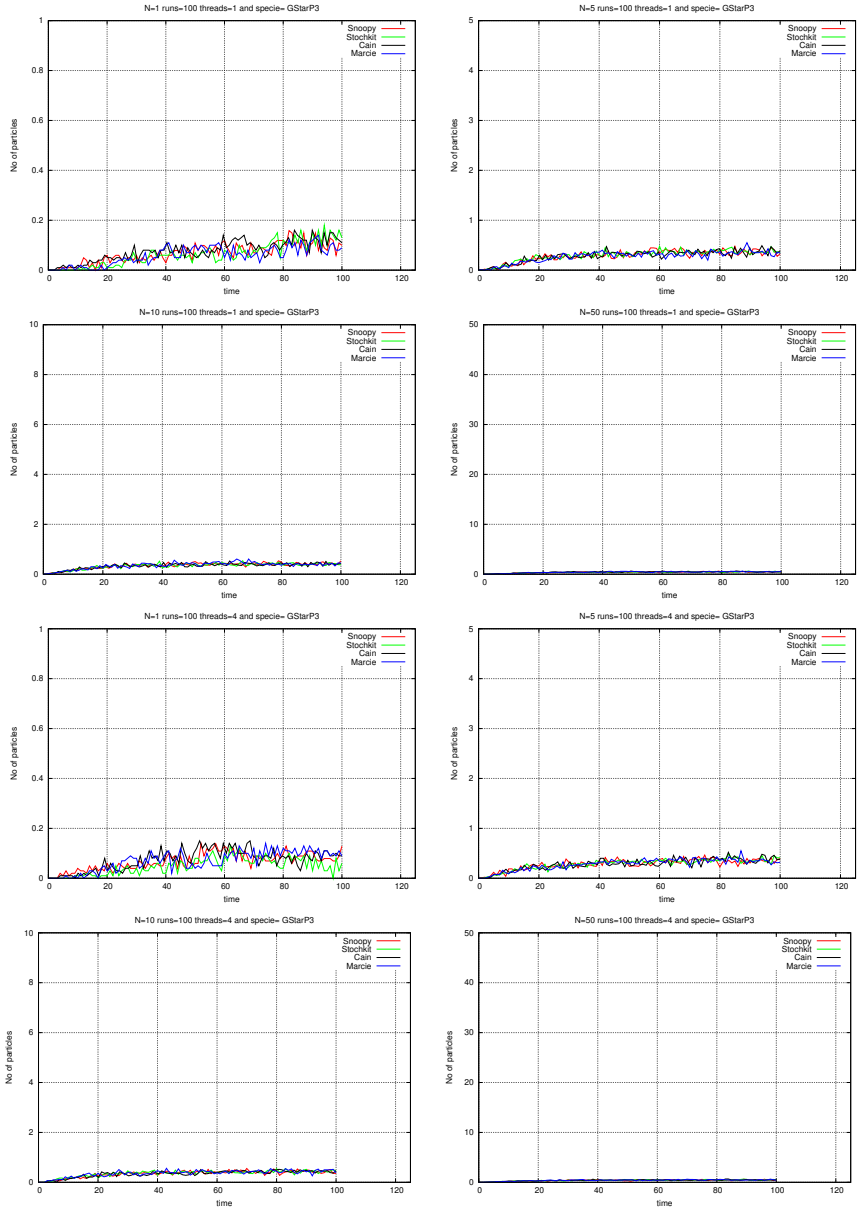

Figure 22: ANGIOGENESIS, accuracy with varying scaling parameter.

A.4.2 Variation with the values of runs

Scaling parameter is kept constant. For ERK, LEVCHENKO and ANGIOGENESIS the values of N are 100, 10 and 5 respectively.

ERK benchmark, specie: MEKPP

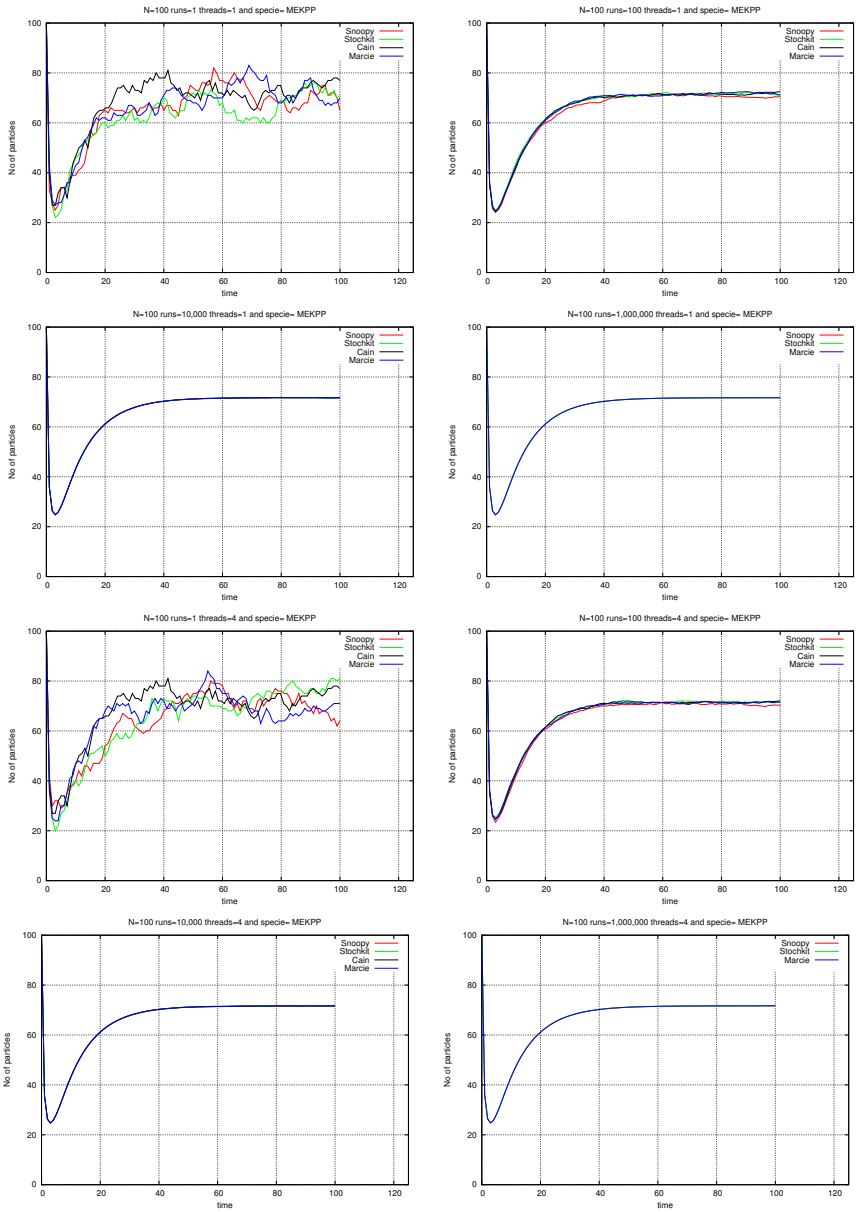

Figure 23: ERK, accuracy with varying runs.

LEVCHENKO benchmark, specie: ERK_MEKPP

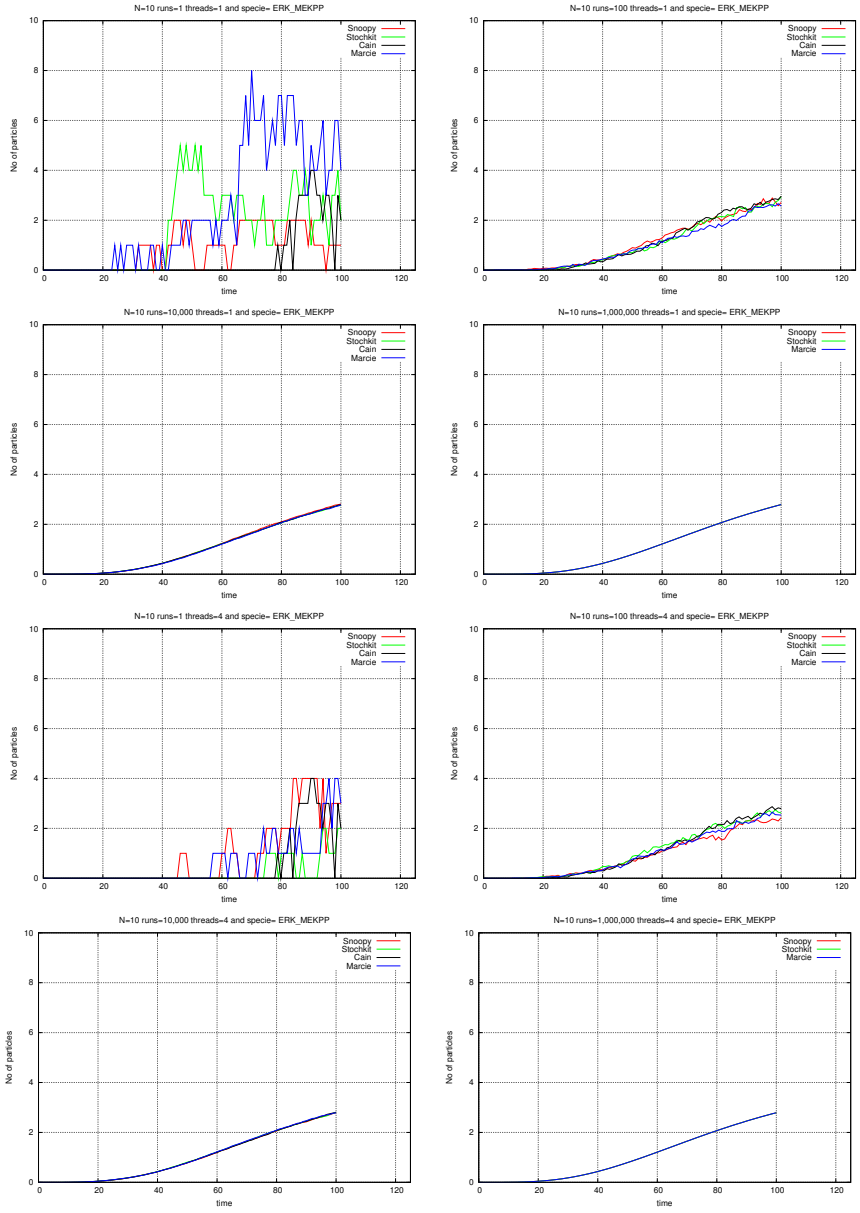

Figure 24: LEVCHENKO, accuracy with varying runs.

ANGIOGENESIS benchmark, specie: GStarP3

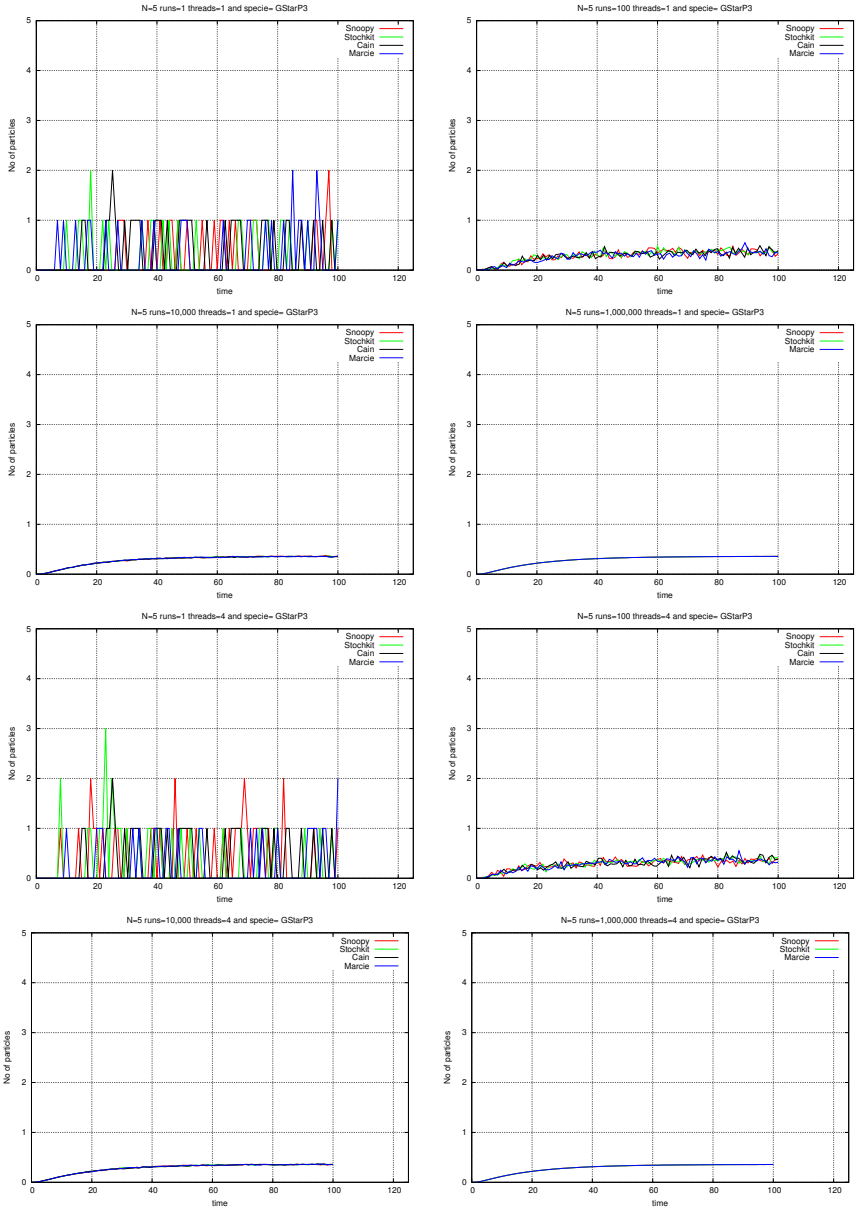

Figure 25: ANGIOGENESIS, accuracy with varying runs.

CIRCADIAN CLOCK benchmark, specie: a

For runs= 1,000,000 the traces are not present because the simulation runtime is greater than 3,600 sec.

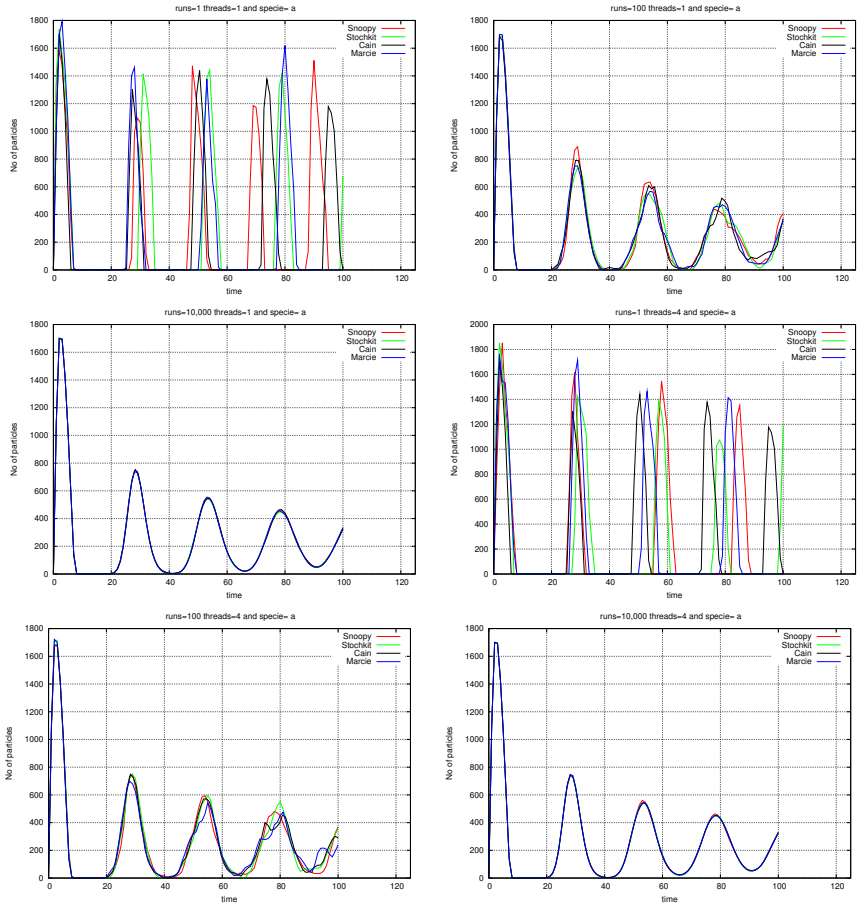

Figure 26: CIRCADIAN CLOCK, accuracy with varying runs.

A.4.3 Variation with the values of threads

We are keeping scaling parameter and number of runs constant. No of runs= 100 and value of N is 100, 10 and 5 ERK, LEVCHENKO and ANGIOGENESIS benchmarks.

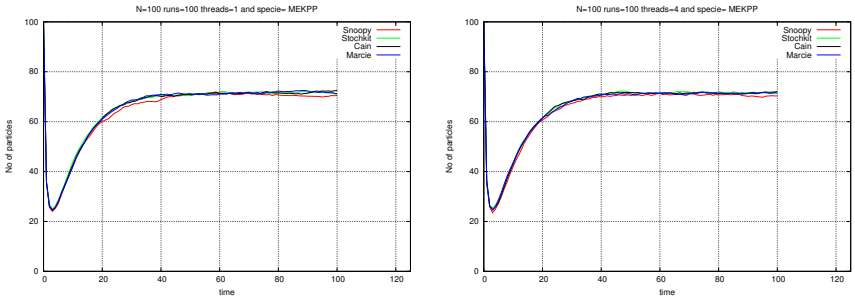

Figure 27: ERK, Accuracy with varying threads.

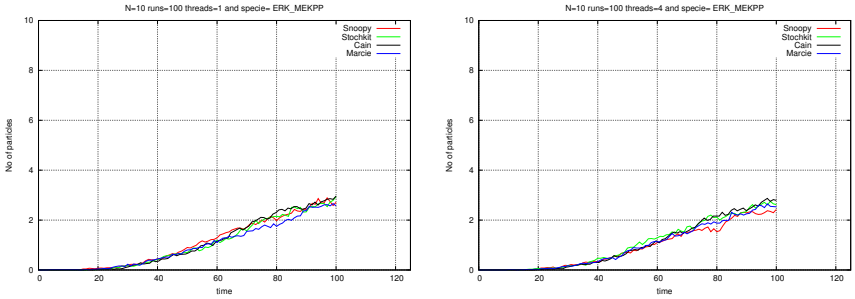

Figure 28: LEVCHENKO, Accuracy with varying threads.

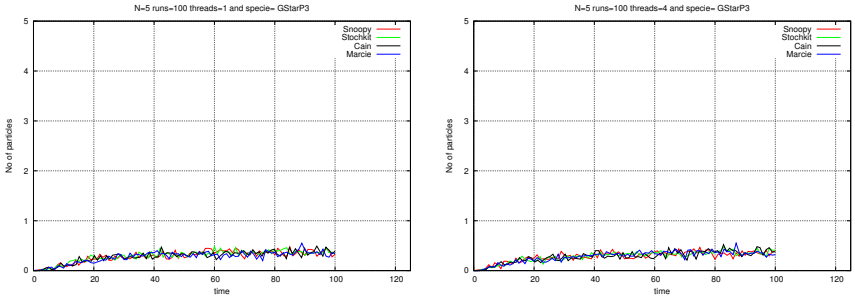

Figure 29: ANGIOGENESIS, Accuracy with varying threads.

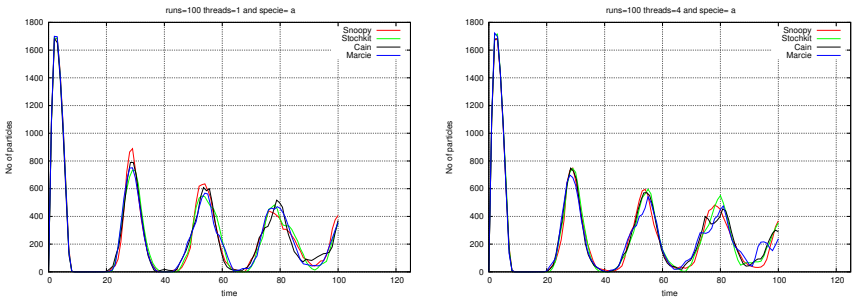

Figure 30: CIRCADIAN CLOCK, Accuracy with varying threads.

A.5 Conclusion

The obtained results matches with the expected results. So we can say that the simulation performed was correct.

B DIZZY

B.1 Incorporation of Dizzy

Dizzy supports SBML level 1. The benchmarks model was written in SBML level 2. Level 2 could not be incorporated in Dizzy. So we developed a SBML level 1 export from Snoopy. ERK benchmark was exported to level 1 and the export was successful. We performed simulation on Dizzy for this benchmark for Dizzy's, however we faced the following limitation with Dizzy:

1. Dizzy does not display the simulation run time. Experiments were performed on Dizzy for ERK benchmark and the simulation run time was recorded by the time command (Linux time command).

2. It does not support multi threading.

3. The tool is very slow in performing simulation. The experiments were carried out and most of the experiments had simulation runtime greater than 3,600 sec. Following can be possible reasons for its slow runtime:

   - The implementation language of Dizzy is Java and the implementation language has an impact on the runtime of the tool. For more information please refer [19]. This may be a reason for its slow runtime.
   - The time measured by the time command includes the writing time of the trace file by Dizzy. This can also be a possible reason.

4. Dizzy GUI does not support plots for more than 20 species at an instance.

B.2 Qualitative comparison of Dizzy

- This tool was developed by Institute for System Biology, Seattle, Washington, US. In order to perform simulation the user manual was referred which is available with the installation file of Dizzy. The tool is available for download on `http://magnet.systemsbiology.net/software/Dizzy/1.11.4/download.html`

- Modelling paradigm: It supports Stochastic and Deterministic models. Few algorithms are:

  – Gillespie stochastic algorithm
  – Gibson-Bruck stochastic algorithm
  – Tauleap-complex

  - Tauleap-simple
  - Deterministic (ODE based) algorithm

- Model class: This discussion is beyond the scope of this report.

- Data exchange formats: Imports and Exports

  - The source file is stored in a .cpp format.
  - Uses a Java Converter to convert the SBML input file to make it compatible with StochKit.
  - The converter accepts the standard version 1 (level 1 and level 2) of SBML and version 2 SBML files.
  - Chemical Model Definition Language (CMDL): is the language understood "natively" by the Dizzy scripting engine.

- Tool features and handling: The tool was found to be easy in handling. The tool has both command line and GUI interface and the all the necessary commands which are used while performing simulations are mentioned in the user manual. The results can be exported to a .csv file.

- Interface: It has both GUI and command line interface. While performing simulation, the GUI interface displays the progress of the simulation int terms of (i.e. how much of the simulation is remaining).It doesnot shows the simulation run time after the simulation is complete.

- Evaluation of results: The simulation results can be exported to a .csv file which can be processed by gnuplot in order to plot the graph. The GUI interface supports plotting function but we cannot plot more than 20 species at a time. It does not support model checking.

- Parallel computing: No.

- Implementation language: It is implemented in Java.

- Platforms: It is supported in Windows, Linux and Mac/OS. Hardware architecture: Only 64 bit for Linux was downloaded and installed.

- License: Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

- Tool version: There are a total of 41 version release for DIZZY with the first version (version 0.0.1) released on 04 August 2003. The latest version of DIZZY is 1.11.4 which was released on 28 September

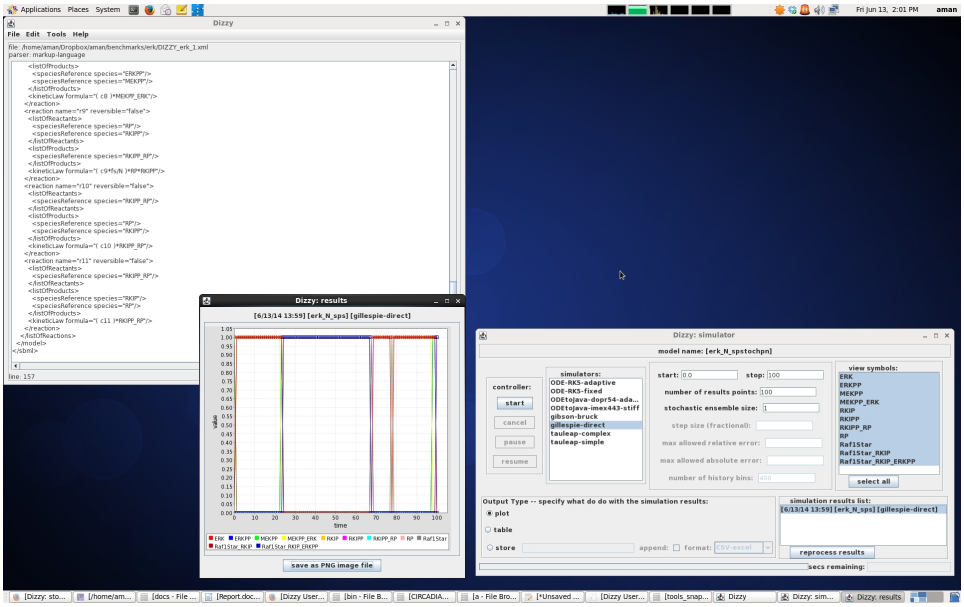

Figure 31: Dizzy Screenshot GUI

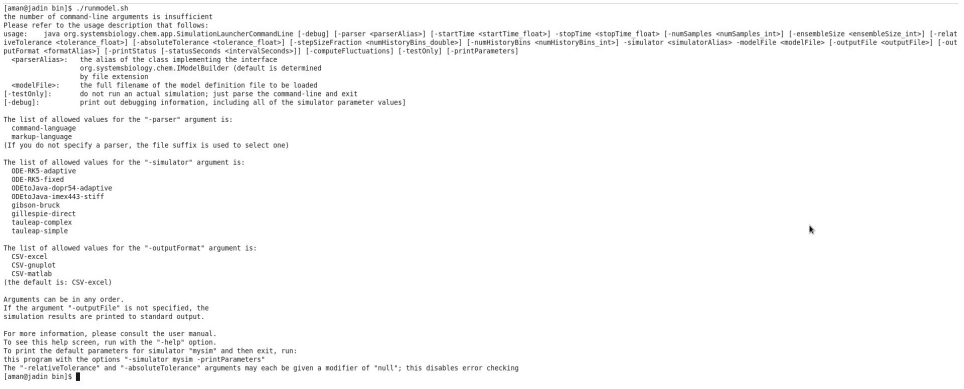

Figure 32: Dizzy Screenshot Command Line Interface

2006. The version of Dizzy downloaded is 1.11.3 which was available on the website `http://magnet.systemsbiology.net/software/Dizzy/1.11.4/download.html`. The Unix version is 1.11.3 which was released on 28 September 2006. This tool was downloaded on 2 April 2014.

- Ease of installation: The link for DIZZY can be found at the SBML website:
`http://sbml.org/SBML_Software_Guide/SBML_Software_Summary#`

`cat_9`
The above link directs to the DIZZY website on
`http://magnet.systemsbiology.net/software/Dizzy/`
The Download tab was clicked and the page got re-directed to
`http://magnet.systemsbiology.net/software/Dizzy/1.11.4/download.html`
The UNIX version was downloaded and the instruction written in the download page was followed.
The installation was easy.

B.3 Simulation on ERK Benchmark

The average runtime and the peak memory consumption recorded for Snoopy for ERK model is given in Table 44 and Table 45 respectively.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	0.6990	0.7930	1.8980	111.5950
	4	–	–	–	–
100	1	0.7710	1.4170	62.4630	†
	4	–	–	–	–
10,000	1	1.3950	62.0140	†	†
	4	–	–	–	–
1,000,000	1	61.3610	†	†	†
	4	–	–	–	–

† runtime $> 3,600$ sec
– means simulation has not been performed

Table 44: Dizzy average runtime (in sec) for ERK.

N	threads	runs			
		1	100	10,000	1,000,000
1	1	66,060	66,980	67,148	78,864
	4	–	–	–	–
100	1	66,708	67,344	77,916	†
	4	–	–	–	–
10,000	1	66,660	77,636	†	†
	4	–	–	–	–
1,000,000	1	66,060	†	†	†
	4	–	–	–	–

† runtime $> 3,600$ sec
– means simulation has not been performed

Table 45: Dizzy peak memory consumption (in KB) for ERK.

B.4 Quantitative comparison

There is no plot for threads= 4 because dizzy does not supports multi threading. The plots are for only ERK benchmark.

B.4.1 Accuracy

We are plotting curves for specie MEKPP. For N=1,000,000 runs=100 Dizzy has simulation time > 3,600 sec.

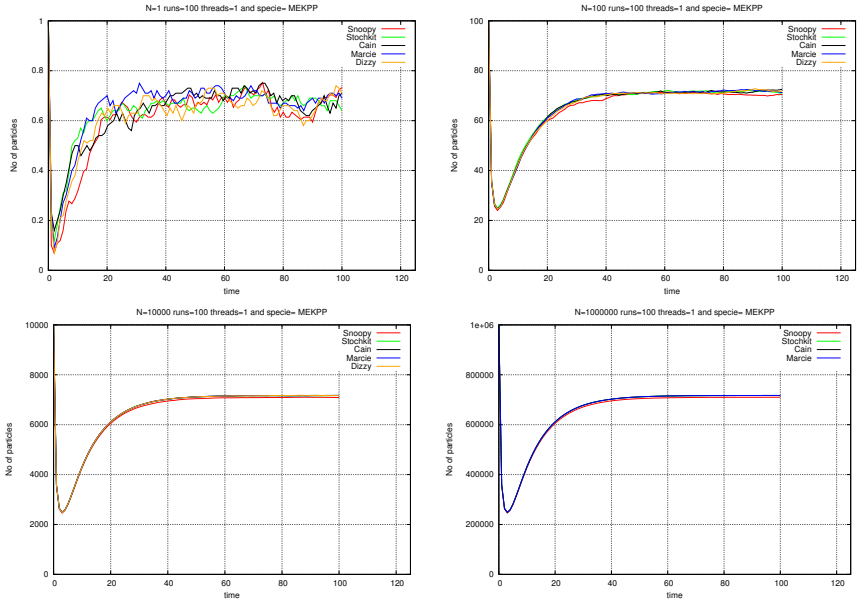

Figure 33: ERK, accuracy with varying scaling parameter.

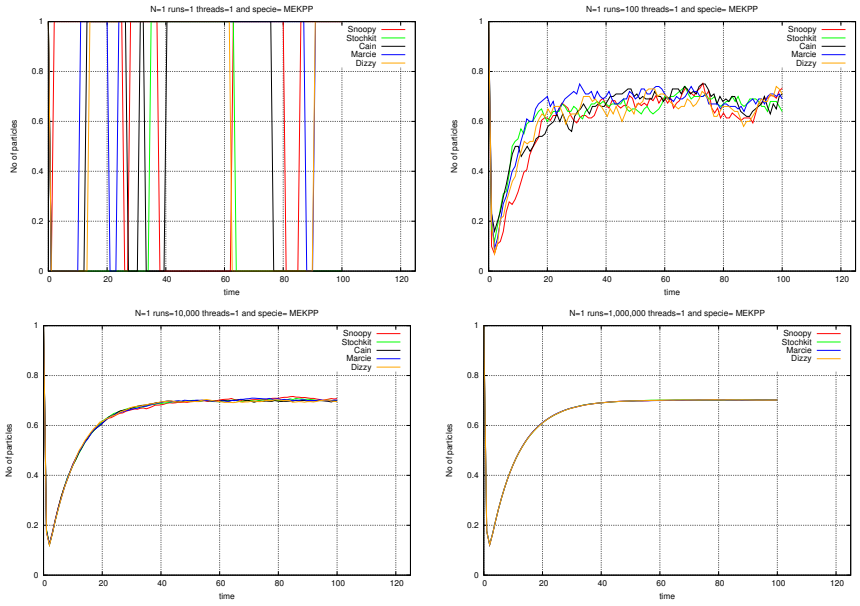

Figure 34: ERK, accuracy with varying runs.

B.4.2 Runtime Comparison

For runtime comparison of the tools refer Figure 35 which is plotted using Table 3 , Table 5, Table 7, Table 9 and Table 44.

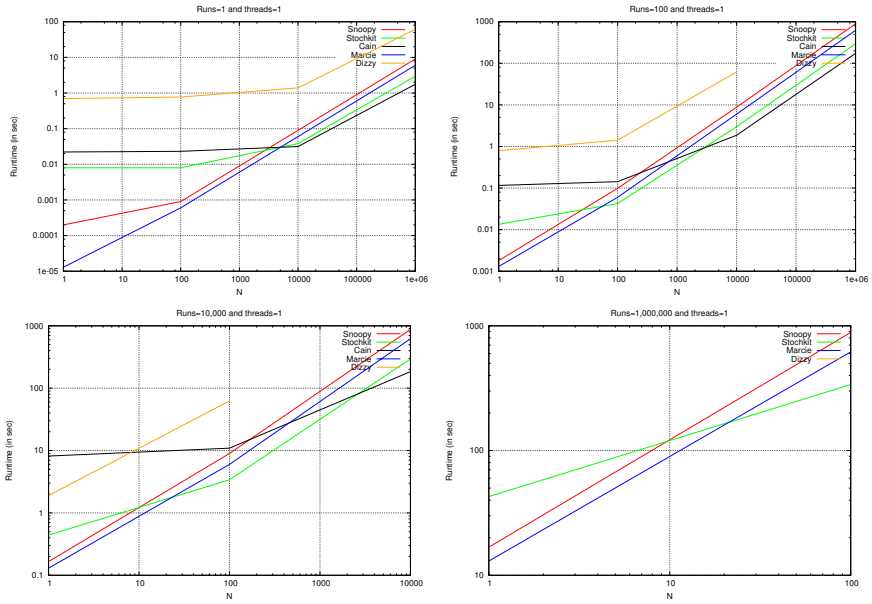

Figure 35: ERK, runtime comparison.

B.4.3 Memory Comparison

For memory comparison of the tools refer Figure 36 which is plotted using Table 4, Table 6, Table 8, Table 10 and Table 45.

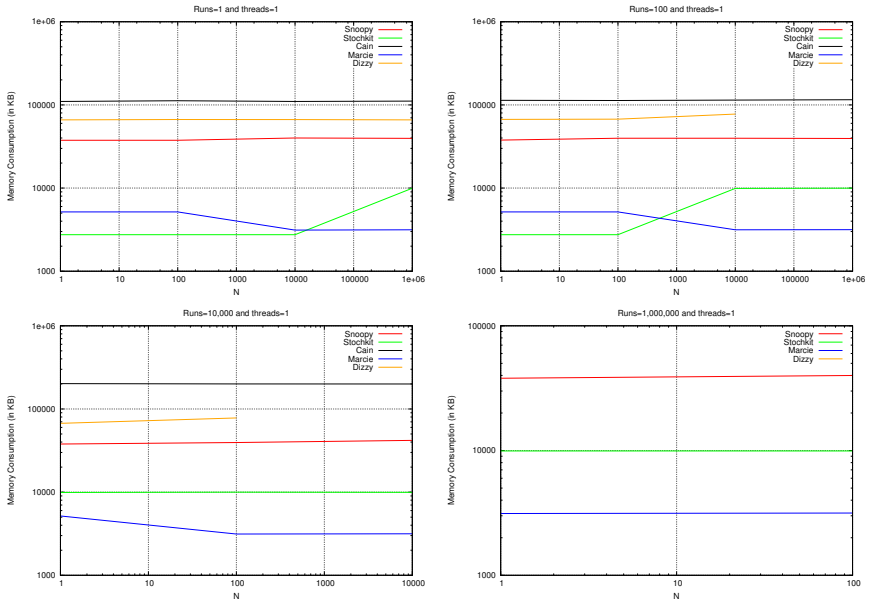

Figure 36: ERK, memory consumption comparison.

B.4.4 Multithreading Coefficient

Dizzy does not support multi threading so there is no comparison.

B.5 Conclusion

Dizzy could not be incorporated with other tools because it was found to be very slow in comparison to other tools and we decided to drop it after comparing it with other tools for benchmark ERK.

C Scripts/Code

The scripts are written in C, C#, C++ and the shell scripts. For generating plots we have GNU plot scripts. Command line tools are operated by the scripts. Scripts can be customized by users. Following are the roles of the script:

1. benchmark.sh - written in Linux shell scripting language. Used by command line tool. For each experiment, which contains 10 trials, it stores the output of the terminal into a .out file. It also stores the runtime and the peak memory usage of the tool into a .csv file. The runtime measured by this script is not the simulation runtime displayed by the tool, it is the time for which the tool runs. We are not interested in this runtime. This runtime is calculated with the help of Linux time command. We are not interested in the runtime in the .csv file, so we develop a parser for the .out file and extract the simulation runtime displayed by the tool. However we are interested in the memory consumption written in the .csv file by this script. The memory consumption is measured using a shell script. benchmark.sh stops the simulation if the runtime is $> 3{,}600$ sec.

2. marcie.sh - written in Linux shell scripting language. Used by command line tool Marcie. It calls Marcie recursively for the different value of scaling parameters, runs and threads. It also calls benchmark.sh to calculate the memory consumption.

3. stochkit.sh - written in Linux shell scripting language. Used by command line tool StochKit. It calls StochKit recursively for the different value of scaling parameters, runs and threads. It also calls benchmark.sh to calculate the memory consumption.

4. dizzy.sh - written in Linux shell scripting language. Used by command line tool Dizzy. It calls Dizzy recursively for the different value of scaling parameters, runs and threads. It also calls benchmark.sh to calculate the memory consumption.

5. marcie_out - written in C#. It is a parser which parses the .csv file and .out file obtained from benchmark.sh for the tool Marcie. It parses the .csv file and reads the memory consumption. It also parses the .out file and reads the simulation runtime displayed by the tool. It reads all the experiment files at once and then writes a .ods file which contains the average simulation runtime and peak memory consumption.

6. marcie_out_cir_clock - written in C#. Same as marcie_out, just an extension for Circadian Clock model.

7. stochkit_out - written in C#. It is a parser which parses the .csv file and .out file obtained from benchmark.sh for the tool StochKit. It parses the .csv file and reads the memory consumption. It also parses the .out file and reads the simulation runtime displayed by the tool. It reads all the experiment files at once and then writes a .ods file which contains the average simulation runtime and peak memory consumption.

8. stochkit_out_cir_clock - written in C#. Same as stochkit_out, just an extension for Circadian Clock model.

9. dizzy_output - written in C#. It is a parser which parses the .csv file obtained from benchmark.sh for the tool Dizzy. Since Dizzy does not display the simulation runtime, the simulation runtime is taken as the runtime in the .csv file created by benchmark.sh. However this is not the correct simulation run time as it includes the writing time of the trace file also, but we cannot do anything because Dizzy does not show the simulation runtime. It reads all the experiment files at once and then writes a .ods file which contains the average simulation runtime and peak memory consumption.

10. runtime_comparison - written in C#. This is used to plot the relative runtime comparison of the tool. The runtime table given in Table 39 (for ERK benchmark) is provided as input and two csv files for each thread with "tab" as datafile separator is made. These files are read by the GNU plot scripts and the curves are plotted.

11. runtime_comparison_cir - written in C#. Same as runtime_comparison, just an extension for Circadian Clock model.

12. standard names.cpp - written in C++. It is used to provide standard naming convention to the traces generated by different tools. It also changes extension of the file to .txt.

13. standard_names_CIR_CLOCK.cpp - written in C++. Same as standard names.cpp, just an extension for Circadian Clock model.

14. standard_names_stochkit.cpp - written in C++. Same as standard names.cpp, just an extension for StochKit tool.

15. standard_names_stochkit_CIR_CLOCK.cpp - written in C++. Same as standard_names_stochkit.cpp, just an extension for Circadian Clock model.

16. copying_files.cpp - written in C++. Copies one trace file for each experiment renamed by standard names.cpp to the specified location in the script. These copied files will be used to plot the accuracy curves.

17. copying_files_CIR_CLOCK.cpp - written in C++. Same as copying_files.cpp, just an extension for Circadian Clock model.

18. gnuplotscripts.cpp - written in C++. This writes 32 text files (for each experiment) containing the GNU plot script to plot the species for a particular benchmark.

19. gnuplotscripts_CIR_CLOCK.cpp - written in C++. Same as gnuplotscripts.cpp, just an extension for Circadian Clock model.

20. gnuplotscripts_dizzy.cpp - Same as gnuplotscripts.cpp, just an extension for Dizzy. Dizzy writes the traces in the lexicographical order. So there is a function provided in this script to map species to its correct location in the trace files.